UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Telecommunications Engineering

Deployment and analysis of a 5G NR radio access network based on Open RAN, using USRPs and OpenAirInterface.

End of Degree Project

Bachelor's Degree in Telecommunication Technologies and Services Engineering

AUTHOR: Mallasen Quintana, Sergio

Tutor: Gómez Barquero, David

Experimental director: CURIESES SANZ, FRANCISCO JAVIER

ACADEMIC YEAR: 2021/2022

Resumen

La quinta generación de comunicaciones móviles cambia el paradigma de la arquitectura de red hacia el uso de funciones virtualizadas, no sólo a nivel de transporte (núcleo de red o core network), sino también a nivel de la red de acceso radio RAN (Radio Access Network), permitiendo el empleo de distintos softwares sobre un hardware genérico para facilitar las actualizaciones y la introducción de nuevas funcionalidades de manera ágil. Así mismo, existen nuevas corrientes que proponen la transcripción del concepto de código abierto a las interfaces y arquitecturas RAN, permitiendo el desarrollo de componentes a terceros y facilitando la interoperabilidad para ganar flexibilidad y eficiencia en los despliegues radio 5G.

La apertura de los protocolos e interfaces de la RAN ha resultado en el concepto de una nueva arquitectura de red denominada Open RAN (O-RAN), que además busca introducir técnicas de IA/ML para el análisis de datos y la toma de decisiones a través de nuevos componentes como el RIC (Radio Intelligent Controller), ejecutándose on-premise, en el edge o en el cloud según las necesidades de latencia y requisitos del caso de uso concreto.

En este trabajo se busca desplegar la RAN de una red 5G basada en el estándar O-RAN, empleando la implementación software de código abierto de OpenAirInterface (OAI) y dispositivos de radio definida por software USRPs, haciendo hincapié en el uso y analizando las posibilidades del novedoso controlador radio RIC introducido en la arquitectura O-RAN.

Resum

La cinquena generació de comunicacions mòbils canvia el paradigma de l'arquitectura de xarxa cap a l'ús de funcions virtualitzades, no només a nivell de transport (nucli de xarxa o core network), sinó també a nivell de la xarxa d'accés ràdio RAN (Radio Access Network), permetent l'ús de diferents softwares sobre un hardware genèric per facilitar les actualitzacions i la introducció de noves funcionalitats de manera àgil. Així mateix, existeixen nous corrents que proposen la transcripció del concepte de codi obert a les interfícies i arquitectures RAN, permetent el desenvolupament de components a tercers i facilitant la interoperabilitat per guanyar flexibilitat i eficiència en els desplegaments ràdio 5G.

L'obertura dels protocols i les interfícies de la RAN ha resultat en el concepte d'una nova arquitectura de xarxa anomenada Open RAN (O-RAN), que a més busca introduir tècniques d'IA/ML per a l'anàlisi de dades i la presa de decisions mitjançant nous components com el RIC (Radio Intelligent Controller), executant-se on-premise, a l'edge o al cloud segons les necessitats de latència i requisits del cas d'ús.

En aquest treball es busca desplegar la RAN d'una xarxa 5G basada en l'estandard O-RAN, emprant la implementació software de codi obert d'OpenAirInterface (OAI) i dispositius de ràdio definida per software USRPs, posant èmfasi en l'ús i analitzant les possibilitats del nou controlador ràdio RIC introduït a la arquitectura O-RAN.

Abstract

The fifth generation of mobile communications changes the paradigm of network architecture towards the use of virtualized functions, not only at the transport level (core network), but also at the radio access network (RAN) level, allowing the use of multiple software over a generic hardware, to facilitate upgrades and the introduction of new functionalities in an agile manner. There are also new trends that propose the transcription of the open source concept to RAN interfaces and architectures, allowing the development of third-party components and facilitating interoperability to gain flexibility and efficiency in 5G radio deployments.

The opening of RAN protocols and interfaces has resulted in the concept of a new network architecture called Open RAN (O-RAN), which also seeks to introduce AI/ML techniques for data analysis and decision making through new components such as the RIC (Radio Intelligent Controller), running on-premise, on the edge or in the cloud depending on latency needs and use case requirements.

In this work we seek to deploy the RAN of a 5G network based on the O-RAN standard, employing the OpenAirInterface (OAI) open source software implementation and USRPs software defined radio devices, emphasizing the use and analyzing the possibilities of the novel RIC radio controller introduced in the O-RAN architecture.

A mis padres.

Contents

I Report

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 2
- 1.3 Objectives . . . 2
- 1.4 Outline . . . 3

2 Framework **5**
- 2.1 3GPP 5G New Radio . . . 5
  - 2.1.1 5G Network Architecture . . . 8
  - 2.1.2 Radio Access Network . . . 8
- 2.2 Open RAN . . . 15
  - 2.2.1 O-RAN . . . 15
- 2.3 Open Air Interface . . . 18
  - 2.3.1 FlexRIC . . . 18
- 2.4 Universal Software Radio Peripherals . . . 20

3 Deployment's Setup **22**
- 3.1 Host PCs . . . 22
  - 3.1.1 Low latency Kernel . . . 22
  - 3.1.2 CPU power management . . . 23
  - 3.1.3 CPU governor . . . 23
  - 3.1.4 KPTI Protections . . . 23
- 3.2 USRP N310 . . . 24
  - 3.2.1 Sampling rate . . . 24
  - 3.2.2 Network configuration . . . 25
  - 3.2.3 Link between the USRPs . . . 27
  - 3.2.4 Time and clock synchronization . . . 27
- 3.3 FlexRIC & OAI Deployment . . . 30
  - 3.3.1 FlexRIC install . . . 30
  - 3.3.2 OAI install and integration . . . 32

4 Methodology **34**
- 4.1 OAI RAN . . . 34
  - 4.1.1 phy-test & noS1 . . . 34
  - 4.1.2 gNB config file . . . 35
  - 4.1.3 RF Simulator . . . 36

4.1.4 Scope . . . 37
4.1.5 T-tracer & Wireshark . . . 38
4.1.6 Spectrum analyser . . . 38
4.1.7 ROMES . . . 40
4.1.8 Ping & Iperf . . . 40
4.2 FlexRIC . . . 43
4.2.1 xApps . . . 43
4.2.2 xApp Challenge . . . 44
4.2.3 O-RAN RIC connection . . . 45

5 Results and Discussion **47**
5.1 OAI RAN . . . 47
5.1.1 Stand-Alone mode . . . 47
5.1.2 Phy-Test mode . . . 52
5.1.3 RF Simulator . . . 54
5.2 FlexRIC . . . 59
5.2.1 Near-RT RIC . . . 59
5.2.2 Available xApps . . . 65
5.2.3 Throughput xApp . . . 69
5.2.4 O-RAN RIC connection . . . 74
5.3 OAI RAN New Version . . . 75
5.3.1 Throughput Benchmark . . . 76

6 Conclusions and Future Work **80**
6.1 Conclusions . . . 80
6.2 Limitations . . . 81
6.2.1 PC performance . . . 81
6.2.2 Server setup . . . 82
6.2.3 OAI and FlexRIC . . . 82
6.3 Future work . . . 83

References **84**

II Annexes

A gNB configuration file **89**

B E2AP available stats list **95**

C Throughput xApp **98**

List of Figures

2.1 Usage scenarios of IMT for 2020 and beyond. . . . 6
2.2 Enhancement of key capabilities from IMT-Advanced to IMT-2020. . . . 7
2.3 The importance of key capabilities in different usage scenarios. . . . 7
2.4 High level architecture of the 5G system. . . . 8
2.5 5G RAN protocol stacks. . . . 9
2.6 Frequency-time domain representation of an OFDM signal. . . . 11
2.7 5G attachment procedure message sequence. . . . 13
2.8 3GPP RAN split 7.2 architecture. . . . 14
2.9 Split-RAN hierarchy, with one CU serving multiple DUs, each of which serves multiple RUs. . . . 14
2.10 O-RAN Architecture. . . . 16
2.11 The E2 interface allows RIC xApps to control RAN functions. . . . 17
2.12 FlexRIC Architecture 19

3.1 USRP N310 front and back panels. . . . 24
3.2 OctoClock-G CDA-2990 front panel. . . . 28
3.3 Deployment setup diagram. . . . 29
3.4 Deployment setup in the Lab. . . . 29

4.1 OAI Scopes explained 37
4.2 R&S *FSW* configuration. . . . 39
4.3 R&S *ROMES* in use. . . . 40
4.4 5G NR Throughput data rate equation in the DL and the UL. . . . 42
4.5 Transport Block Size determination algorithm 42

5.1 R&S *ROMES 4* showing the 5G NR spectrum. . . . 49
5.2 R&S *ROMES 4* showing the 5G NR scanner. . . . 49
5.3 R&S *ROMES 4* showing the SA BCH info. . . . 50
5.4 R&S *ROMES 4* showing the SA SIB1 info. . . . 51
5.5 Execution with phy-test mode over USRPs 52
5.6 R&S *FSW* spectrum analyzer showing the gNB DL capture. . . . 53
5.7 *Wireshark* OAI Tracer's capture showing a `phy-test` idle sequence. . . . 54
5.8 *Wireshark*'s OAI Tracer capture showing a MIB message. . . . 55
5.9 *Wireshark*'s OAI Tracer capture showing two MAC messages' details. . . . 55
5.10 *Wireshark* OAI Tracer's capture showing a UL ping message. . . . 56
5.11 RF Simulator theoretical throughput vs. measured throughput 58
5.12 E2AP Setup and RIC subscription sequence 60
5.13 E2AP Setup packet details 61
5.14 RIC Subscription packet details 62

5.15 RIC Indication packet details . . . 63
5.16 RIC and iApp Indication sequence . . . 64
5.17 iApp Indication packets details . . . 64
5.18 xApp communication over the E42AP interface . . . 67
5.19 All layers' throughput xApp showing different Iperf runs. . . . 70
5.20 Throughput xApp showing three DL UDP iperf test with 1M, 10M and 50M of bandwidth. . . . 71
5.21 Throughput xApp showing three DL TCP iperf test with 1M, 10M and 50M of bandwidth. . . . 72
5.22 Throughput xApp showing a UL TCP iperf test. . . . 73
5.23 OAI RAN newer version's logs . . . 75
5.24 OAI RAN newer version's scopes . . . 76
5.25 USRP theoretical throughput vs. measured throughput . . . 77
5.26 RFSIM vs USRP throughput . . . 78

List of tables

3.1 USRP N310 common sampling rates. . . . 25

4.1 MCS index table 1 for PDSCH (Selection) . . . 43

5.1 RF Simulator UDP throughput benchmark . . . 57
5.2 Over-The-Air UDP throughput benchmark . . . 77

Part I

Report

Chapter 1

Introduction

1.1 Context

Communication is an inherent part of humankind; in fact we are the only known species to develop formal languages to share and store information. Therefore, all over the history, there has always been an interest in revolutionizing the way we communicate, from the invention of written languages, the Gutenberg printing press, radio and TV broadcasts, and more recently, digital communication in its many shapes and forms.

The Third Industrial Revolution -also known as Digital Revolution- came as technology shifted from analog to digital electronic computing, proliferating not only the individual machines but also the networks they communicate with, as the Internet and the World Wide Web rose. The mass production and widespread of microprocessors, smartphones, computers and its interconnectivity evolved into the new technologies, industries and societal patterns that define the Fourth Industrial Revolution -or Industry 4.0- that we are experiencing today.

It is well known that mobile communications technologies are being designed to meet each decade's needs, starting with the first generation of wireless telephone technology in the 80s, which was capable of voice-only analog communication. Then came the first digital technology in the 90s, standardized in Europe as GSM, that could also send and receive SMS text messages. As Internet and data transmission became more relevant in the new millennium, there was the need for a new global standard and so 3GPP was established in 1998 to work on the following generations. The 3G technologies allowed for data and voice communication over the same network using circuit-switching.

In the 10s decade, smartphones and mass-consumer electronics became widespread and more connected to the Internet, so 4G technology switched to an IP-packet based routing that integrated fixed and mobile broadband networks and increased data transmission rates. It is the first generation to have a single world-side standard: Long Term Evolution (LTE).

Entering the 2020s decade, new services called for a fifth generation that fulfilled its requirements for high data rates, reduced latency and increased system capacity. Mobile gaming, mass media streaming platforms, video calls, Internet-of-Things and autonomous vehicles are some applications for which 5G is designed.

It is important to understand that 5G implies much more than a generational upgrade in bandwidth

as it involves the transformation of the access network to make it more like a modern cloud. Some technology trends such as software-defined networking and open-source software will lead to an innovative and flexible RAN. The new demands that 5G has foreseen include more traffic volume, many more concurrent devices with diverse service requirements, better quality of user experience (QoE) and better affordability by reducing costs. [1]

1.2 Motivation

Technology and connectivity is growing its presence in many sectors where it used to be more niche. Industry 4.0 is making use of ubiquity of microprocessors and it is introducing sensors, communication and intelligence to every step or process of its chain of value. On the other hand, consumers demand higher streaming rates for mobile real-time gaming and media.

This is creating new demands for the Radio Access Network which did not evolve as much as the core in the past. Private 5G networks, new ways of covering areas depending of its use and the need for higher bandwidths and spectrum increases the radio complexity.

In the past, even though mobile technology has been publicly standardized, radio hardware used to be vendor specific, closed and privative; "black boxes" that made up the network. This obscurity made it very inefficient to combine modules from different sellers. Thus, it did not allow for interoperability between vendors or flexibility in its deployment.

Recently, the open-source code concept has migrated towards the hardware components of computers and machines. A good example of it is the RISC-V instruction set architecture, a global initiative for open-source CPU chips that the EU commission is playing close attention to and financing. The introduction of this new paradigm to the telecommunications field is evolving the traditional RAN towards a open, flexible and virtualized network known as Open RAN. This concept aims to allow for a multi vendor deployment to use interchangeable hardware and software with open interfaces.

The main project in this direction towards openness and intelligence is the O-RAN standard, that defines the new E2 Interface and RAN Intelligent Controller (RIC). This standard is open to third party applications (xApps) for added features. A recent implementation of this standard is FlexRIC. To deploy a network like this, the OpenAirInterface (OAI) software is frequently used, together with USRP prototyping RF radio boards.

1.3 Objectives

The aim of this BSc. thesis is to deploy a 5G RAN using Open Air Interface open-source code, with the Open RAN implementation by Eurecom. This work is done inside the Valencia Campus 5G initiative at Mobile Communications Group (MCG), in the Institute of Telecommunications and Multimedia Applications (iTEAM), of the Polytechnic University of Valencia (UPV).

In order to achieve the main goal, the next specific objectives are determined:

- Study the 5G NR network architecture, particularly the radio access network's architecture, as well as how O-RAN modifies the RAN.

- Configure and deploy a OAI 5G RAN, with FlexRIC, using USRPs as the radio heads.
- Measure the OAI 5G RAN performance.
- Analyse how O-RAN E2 protocol works and the analytics that FlexRIC provides.
- Inspect the available FlexRIC xApps and examine the information they provide.
- Understand how xApps are built and design a new xApp that provides different information.

1.4 Outline

This section provides an overview of this project's structure, exposing the contents of each chapter, to guide the reader through the document.

Chapter 2. Framework. This chapter gives a general reference of all the concepts relevant for this work, from theoretical explanations of 5G NR and the O-RAN standard to a description of the software and hardware utilized.

Chapter 3. Deployment's Setup. In this chapter, the setup employed for the RAN deployment is explained, highlighting the hardware's configuration and software's install.

Chapter 4. Methodology. This chapter provides an in-depth description of the OpenAirInterface and FlexRIC software implementation's options, features and tools, along with third-party applications and hardware devices used for the development and execution of this project.

Chapter 5. Results and Discussion. This chapter exposes the OAI RAN and FlexRIC deployment results, along with the different tests performed, and discusses the information obtained.

Chapter 6. Conclusions and Future Work. In this chapter the conclusions for this project will be exposed, indicating the limitations encountered and providing references for future work in this field.

Chapter 2

Framework

This chapter will first give an overview of 5G NR, showcasing the different use scenarios and key capabilities that define this technology. It is followed by an introduction to its architecture and an in-depth explanation of the RAN. Next, the Open RAN concept and O-RAN Standard are discussed, focusing on the E2 Interface. Finally, a description of the main software, OAI and FlexRIC, and hardware elements, USRPs, is given.

2.1 3GPP 5G New Radio

Back when 3G was being envisioned, there was the need for an association that united different telecommunications organizations around the globe to develop a single mobile communications technology to be deployed by network operators. The result of this vision was the 3rd Generation Partnership Project (3GPP), created in 1998. The 3GPP's aim is to produce Technical Specifications and Technical Reports that form a mobile technology Standard. 3GPP standards are structured as Releases.

Releases 15, staged roll-out in late-2017 and early-2018; Release 16, completed by 2020; and the recently published Release 17 in September 2022, make up the 3GPP 5th Generation (5G), also referred to as New Radio (NR). The fifth generation of mobile networks is the result of the increased demand for wireless communications form many different targets and applications, scenarios. It evolves from LTE with a broader scope (point of view) to fulfill the many needs that industries and consumers plan for the near future.

In 2015, the ITU-R issued the requirements for 5G networks, devices and services in the IMT-2020 recommendation M.2083 (09/2015) for 2020 and beyond [1]. Back then, some of the perceived user and application trends were:

- Supporting very low latency and high reliability for human and machine-to-machine communication.
- Supporting high user density by the increase of users and devices.
- Maintaining high quality at high mobility.
- Enhanced multimedia services in areas beyond entertainment.

- Internet of Things

Therefore, for the first time in a mobile communications technology, three usage scenarios where envisaged:

- **Enhanced Mobile Broadband *(eMBB)***: human-centric use cases for access to multi-media content, services and data. This scenario tackles data rate requirements for hot-spot and wide-area coverage depending on user density and mobility situations.
- **Ultra-reliable and low latency communications *(URLLC)***: remote real-time high-precision control such as remote surgery, industrial manufacturing robots… This use case has strict demands on throughput, latency and availability.
- **Massive machine type communications *(mMTC)***: huge numbers of IoT connected devices transmitting low volume of not real-time data, using low cost and low power hardware.

This use cases are usually represented in a triangle shape like that of Fig. 2.1, where each application is represented as more or less demanding on each case.

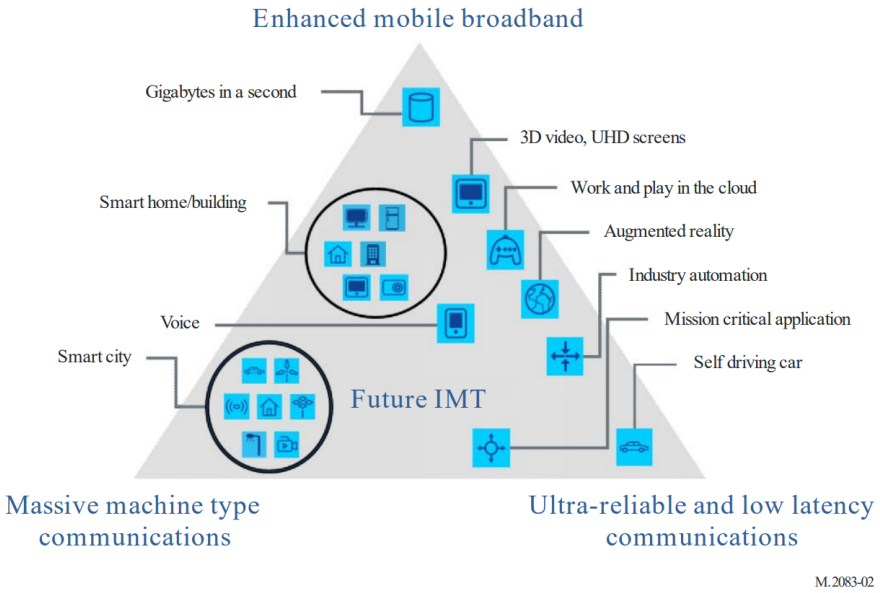


Figure 2.1: Usage scenarios of IMT for 2020 and beyond. [1]

Therefore, to achieve these goals IMT-2020 establishes eight key capabilities that this generation should reach, which are represented in Fig. 2.2.

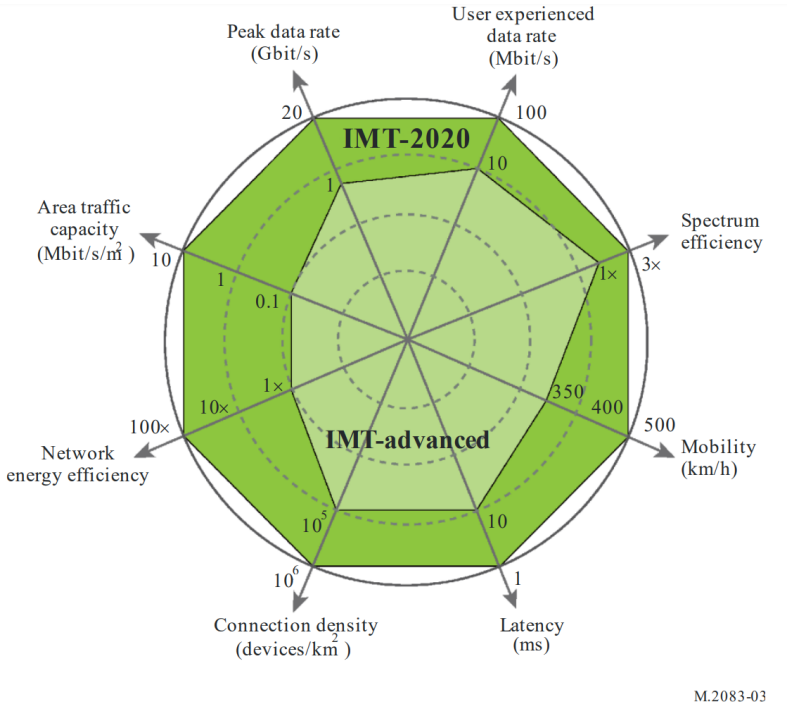


Figure 2.2: Enhancement of key capabilities from IMT-Advanced to IMT-2020. [1]

In this octagon representation, each use case benefits of different key capabilities and some are essential for it to work properly. The Fig. 2.3 illustrates each usage scenario parameter's importance.

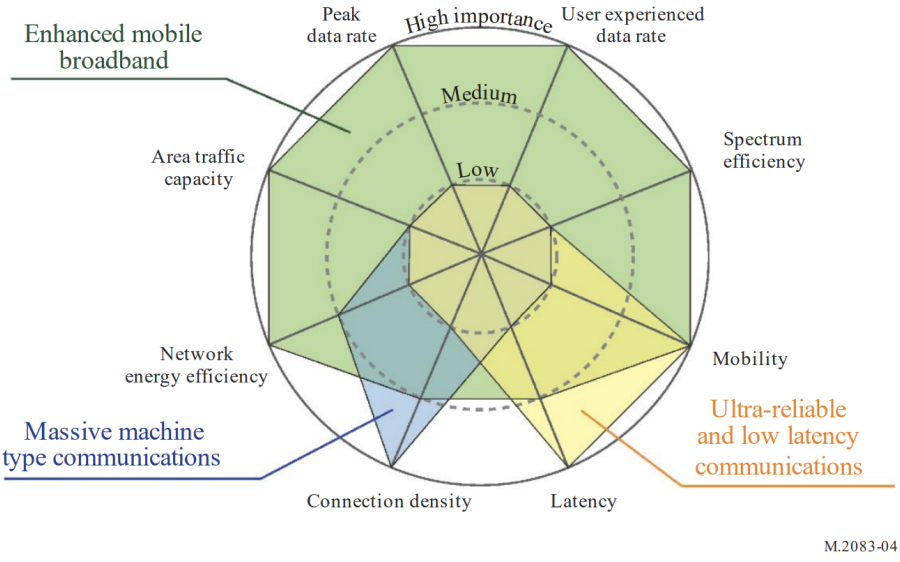


Figure 2.3: The importance of key capabilities in different usage scenarios. [1]

2.1.1 5G Network Architecture

At the high level, the network architecture of a mobile communications system is comprised of a mobile device or User Equipment (UE) that connects via a radio link to a Base Station (BS). This BS handles the radio spectrum to signal the UE to attach, transmit, receive whatever information is needed, either control or data. These elements conform the Radio Access Network. Several BSs are connected to a Core Network (CN) that provides Internet connectivity, ensuring it meets Quality of Service (QoS) requirements, it tracks users in motion to provide uninterrupted service by switching BS, and controls users access and subscribers usage for billing.

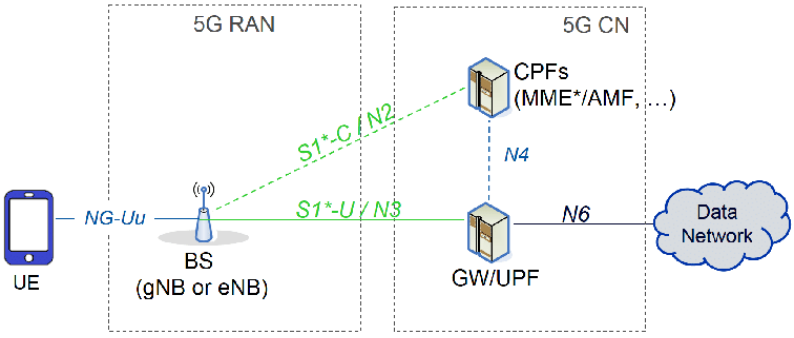


Figure 2.4: High level architecture of the 5G system. [2]

This architecture is then standardized for each Generation to introduce updates and new features. For example, in 4G LTE the BS are called evolved Node-B (eNB) and the CN is called Evolved Packet Core (EPC), and in 5G NR they are called Next Generation Node-B (gNB) and the Next Generation Core (NG-Core).

As an evolving technology and due to time constrictions to release some part of this new standard, 5G is envisioned with two operating modes to allow for a slow transition from LTE to NR. Initially, with theN5G Core still not ready, the Non-Standalone was released. This option consists of a 4G and 5G RAN over 4G's EPC, which meant that a faster deployment could be done. In this mode, control plane traffic between the UE and Core goes through the 4G eNB and the 5G gNB is there to provide a data-rate and capacity boost. Later, the Standalone mode consisting of a 5G RAN and 5G's NG-Core routes both control and user plane traffic exclusively over a New Radio network, making it independent of 4G.

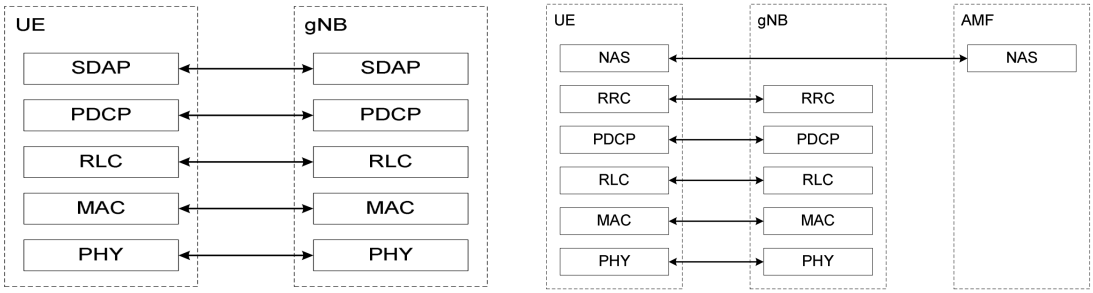
2.1.2 Radio Access Network

The main elements that comprise a 5G RAN or NextGeneration-RAN are Next Generation NodeB (gNB) and 5G-capable User Equipment (UE). To describe the RAN we must understand the role that a gNB plays in it. First, each gNB establishes a RF channel for when a UE is active. Second, it establishes a control plane connection between the UE and the CN, and routes traffic between them to enable authentication, registration and mobility tracking. Third, the gNB creates a user plane connection between each active UE and the CN and forwards control and data packets. While a UE is connected it can physically move, loosing connection with one gNB, that will handle the handover to another gNB. The process by which a UE detects a gNB, establishes a channel and registrates with the CN is called Registration (Attachment in LTE terminology).

2.1.2.1 RAN Layers

Just as in many other telecommunications fields, the RAN is organized on a protocol stack where each entity is designed for a specific task and communicates with its upper and lower layers. This allows for specific information to be sent and receive between the same layers at the gNB and UE. There's one protocol stack for the user plane and another for the control plane, although both share many similarities.

The lower entity of the stack is 5G Layer 1 or Physical layer. Then there is 5G Layer 2 which comprises the MAC, RLC and PDCP sub-layers. Finally, the control plane includes 5G Layer 3 or RRC layer.



(a) User plane **(b)** Control plane

Figure 2.5: 5G RAN protocol stacks. [3]

Each layer's tasks is similar to LTE and it's described in [3]. The following is an explanation of each layer's most relevant features.

PHY Sitting at the bottom of the stack, closer to the analog RF equipment, the Physical layer performs the coding and modulation of the transport channels into physical channels that are used to carry data over the radio interface, as well as error correction.

Physical channels include PDSCH, PDCCH and PBCH for the downlink; and PRACH, PUSCH and PUCCH for the uplink.

- Physical Downlink Shared Channel, carries a variety of data such as user data, UE-specific higher layer control messages, system information blocks and paging. It uses adaptive modulation and flexible coding scheme dependent upon the link conditions.
- Physical Downlink Control Channel's primary function is scheduling the downlink and uplink transmissions on the PDSCH and PUSCH, using a fixed QPSK modulation.
- Physical Broadcast Channel provides periodically with the Master Information Block and helps with synchronization.
- Physical Random Access Channel that handles the first message from the UE to the gNB and the attachment procedure.
- Physical Uplink Shared Channel is used to carry data as its counterpart the PDSCH. It also has a flexible format.

- Physical uplink control channel carries the uplink control data.

Some of the main features of this layer in comparison to LTE include a wide frequency spectrum from the MHz to mmWave bands, OFDM waveforms, scalable numerology with flexible sub-carrier spacing (SCS) from 15kHz to 120kHz, and LDPC advanced channel coding.

One of the main procedures of the PHY layer includes Cell Search: UE receives the Primary and Secondary Synchronization Signal (PSS and SSS) together with the PBCH in what is known as the Synchronization Signal Block (SSB). This allows the UE to acquire time and frequency synchronization for a cell and detect the Master Information Block (MIB). The MIB includes the parameters required to decode the System Information Block Type 1 (SIB1) transmitted over the PDSCH from the RRC, that provides all necessary details to access the network.

MAC The Medium Access Control sub-layer is responsible for all real-time scheduling decisions about what frames are transmitted when, and it does the mapping between logical and transport channels, deciding how such information should be carried. This is called data transfer and radio resource allocation and it's the job of MAC, so that upper layers do not need to worry about when and how data is sent over the air interface. Therefore, it multiplexes and de-multiplexes Service Data Units (SDU) onto Transport Blocks (TB).

An important MAC Procedure to mention is Random Access (RA), that handles the initial UE signal to the gNB when it is powered on. Its main purpose is to achieve UL synchronization between UE and gNB, be assigned a Radio Network Temporary Identifier (RNTI) and granted UL resources, and to establish a Radio Resource Control (RRC) Connection to make possible to communicate between UE and base station.

RLC The Radio Link Control's job is to transfer upper layer Protocol Data Units (PDU) onto logical channels, and it can be configured in one of three transmission modes:

- Transparent Mode (TM): It's the simplest of all as it does not add any header or segment SDUs, making the RLC layer 'transparent'.
- Unacknowledged Mode (UM): It segments SDUs into PDUs and adds a header to them. In reception, it reassembles the segments to deliver complete SDUs to upper layers. When receiving, it may reorder or discard segments based on sequence number.
- Acknowledged Mode (AM): This mode has the important feature of error correction through ARQ on top of the UM mode. In reception it sends feedback on STATUS PDUs to tell the transmitting entity to re-transmit PDUs.

So, the main services and functions depend on the transmission mode, here are some of them:

- Sequence numbering (UM and AM)
- Error correction through ARQ (AM)
- Segmentation (AM and UM) and re-segmentation (AM) of RLC SDUs
- Duplicate detection (AM)
- RLC SDU discard (AM and UM)

PDCP It stands for Packet Data Convergence Protocol and provides the upper layers with services to transfer user and control plane data, IP-packet header compression/decompression, ciphering/deciphering and integrity protection. Apart from that, it also has functions for duplicate discarding, out-of-order delivery, reordering and in-order delivery, and maintaining sequence numbers [4].

2.1.2.2 Radio Frame Structure

A key aspect of 5G that has given it the New Radio designation is how the radio resources are allocated and structured in the time domain and frequency spectrum.

A crucial element of 4G and 5G standards is the use of Orthogonal Frequency-Division Multiplexing (OFDM) as the digital modulation technique, extended for multi-user channel access with Orthogonal Frequency-Division Multiple Access (OFDMA). The physical radio resources of an OFDMA scheme are divided in time, into symbols, and in frequency, organized into sub-carriers following the OFDM modulation.

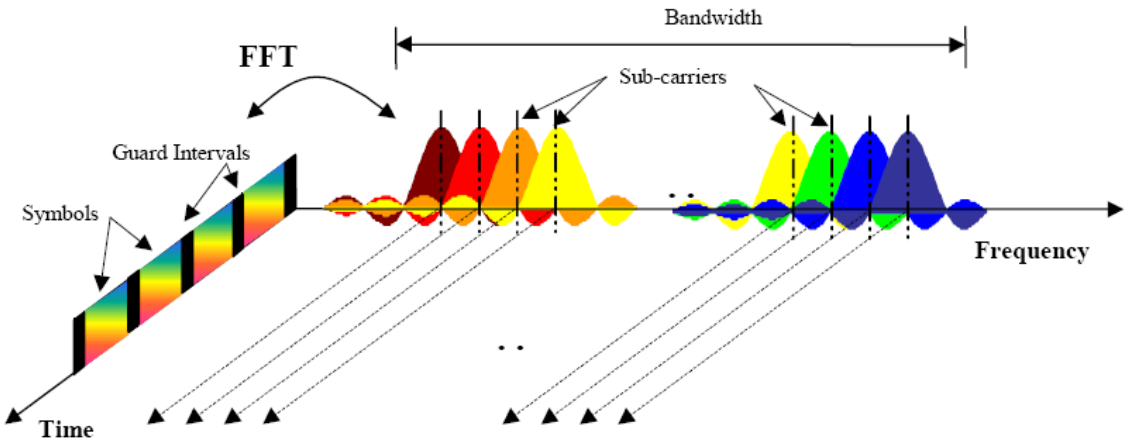


Figure 2.6: Frequency-time domain representation of an OFDM signal. [5]

The smallest unit of resource is one sub-carrier for one symbol duration, called Resource Element (RE). Twelve sub-carriers consecutive in frequency, twelve RE, define a Physical Resource Block (PRB). The 5G signal bandwidth is measured in PRBs of a specific Sub-Carrier Spacing (SCS) so, for example, in Eq. 2.1 106 PRBs with 30kHz SCS take up 38.16MHz of bandwidth.

$$12\,[subcarriers] * 30kHz\,[SCS] * 106\,[PRB] = 38.16MHz \tag{2.1}$$

5G NR can use Frequency (FDD) or Time Division Duplexing (TDD) to create the UL and DL channels. In this work, TDD is used. With TDD, the time domain symbols are grouped into slots (usually 14 symbols per slot). The different SCS available are called numerology (μ), which is introduced in 5G to allow for different spacing at different frequency bands for more flexibility, and also affects the OFDM symbol duration. The Radio Frame structure depends on the numerology, but its length is always 10ms. As the slots have a fixed length of 14 symbols, each numerology allows for a different number of slots in it.

The numerology used in this project is $\mu = 1$ or 30kHz SCS, which defines an OFDM symbol duration (with cyclic prefix) of 35.68 micro-seconds. The 14 symbols in a slot means that the slot's duration is 0.5 mili-seconds. Then, in one 10 mili-seconds radio frame we can fit 20 slots.

With TDD, each slot is configured for DL, UL or in a flexible way, assigning some symbols as DL and others as UL, or serving as a gap. Each type of slot is represented as D, U or F. A periodic pattern of DL and UL slots is defined, the period's duration of which is a sub-multiple of a radio frame, for example 5ms. In those 5ms, 10 slots can be allocated as D, U or F slots to create the pattern, for example DDDDDDDFUU (7D, 1F, 2U). The radio frame will be filled with this pattern in a periodic way, so in this example the frame would be DDDDDDDFUU DDDDDDDFUU (two 10 slot periods adding to 20 slots).

2.1.2.3 UE Attachment Procedure

A 5G SA architecture includes a 5G UE, gNB and NG-Core. For a UE to register to the network a Registration/Attach message flow must occur between itself, the gNB and the 5G Core Access and Mobility Management Function (AMF). In this project the RAN will be deployed without a Core so I will mainly detail the flow for Over-The-Air (OTA) messages (between the UE and gNB).

In broad terms, the 5G SA Attachment fill first attains DL sync via the Synchronization Signal Block (SSB) by detecting the Primary and Secondary Synchronization Signal which are periodically broadcasted over the PBCH. In the SSB is located the Master Information Block (MIB) that includes the parameters required to decode System Information Block 1 (SIB1). The SIB1 is transmitted over the PDSCH and includes many of the key parameters of the gNB and radio channel such as the TDD slot configuration, bandwidth, cell identification...

Once the UE has decoded the information above, it starts the Random Access (RACH) procedure to achieve UL sync and start communication with the RRC by sending RACH Message 1. The RRC in the gNB sends an RA response indicating a temporary Radio Network Temporary Identifier (RNTI) and the Message 3 allocation. The UE then sends the scheduled RRC Setup Request Message 3 with this RNTI with a UE-identity random number to deal with contention resolution. The gNB answers with a RRC message with said random number and the MasterCellGroup Information.

The UE then sends a RRC Setup Complete message and a Registration Request for the AMF at the core. Finally, the UE and AMF start registering over several messages until a PDU Session is established.

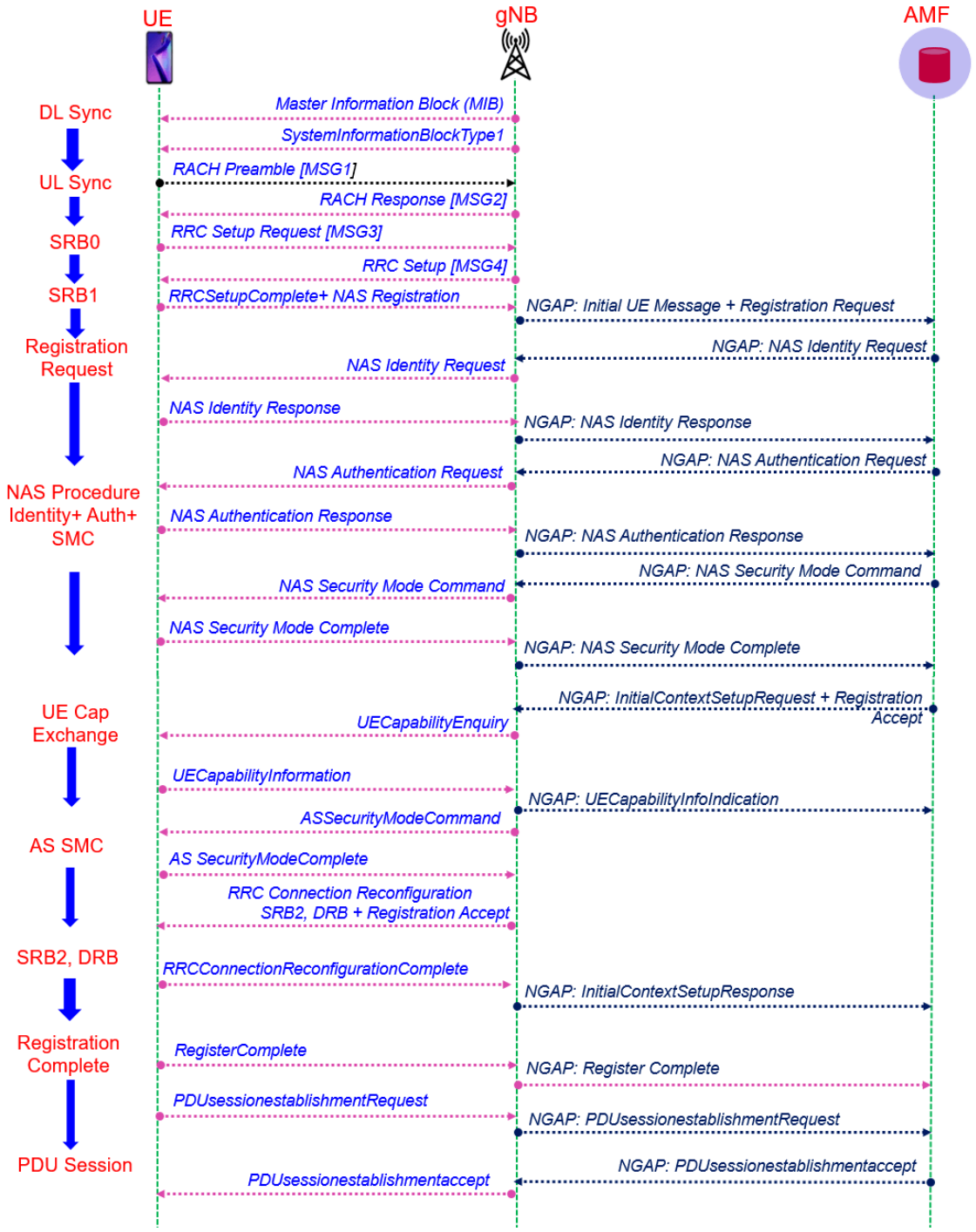


Figure 2.7: 5G attachment procedure message sequence. [6]

2.1.2.4 Split RAN

The dominant deployment has traditionally been to run the entire RAN protocol stack shown in Fig. 2.5 in the physical base station. Going forward to 5G NR, the 3GPP standard Release 15 has been updated to allow it to partition between several physical elements, making use of Network Function Virtualization and Software Defined Network.

This "split" is done across centralized and distributed locations into three logical nodes: the Central Unit (CU), the Distributed Unit (DU) and the Remote Radio Unit (RRU). Release 15 allows for multiple split-points, but Split 7-2, displayed in Fig. 2.8, is a common partition. [7]

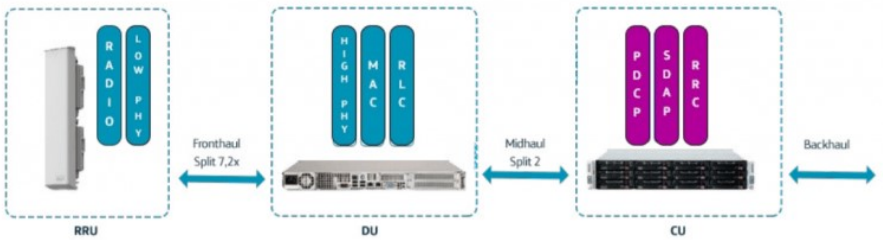


Figure 2.8: 3GPP RAN split 7.2 architecture. [8]

This results in a RAN configuration where a single CU running in the cloud can serve several DUs (in an edge cloud environment), and each of them serving multiple RRUs. This way a gNB is delocalized both in physical and virtualized entities allowing for a cost-efficient, scalable, flexible RAN. Because of this split, latency critical services like real-time scheduling decisions by the MAC layer, in the DU, need to be near (within 1ms) to its RRUs; while near real-time decision making by the RRC, in the CU, can be further away.

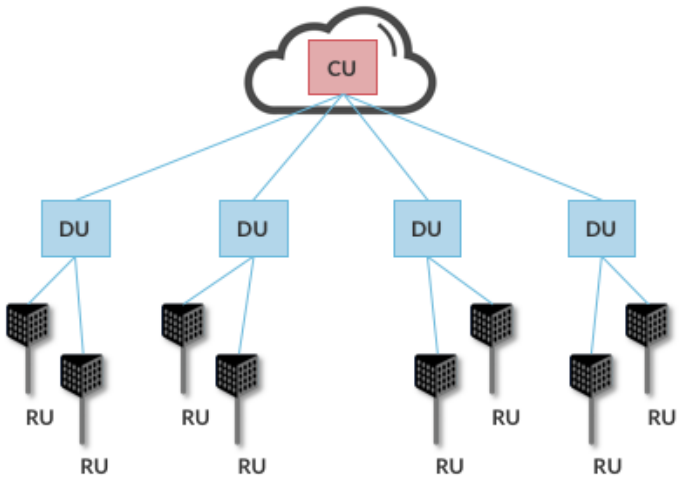


Figure 2.9: Split-RAN hierarchy, with one CU serving multiple DUs, each of which serves multiple RUs. [7]

Although it is common to physically locate the DU and RU in the same cell tower, small cell configurations such as those that come with mmWave in 5G will benefit of a single DU serving multiple RRUs.

2.2 Open RAN

Although 3GPP's open specifications describe all the elements that implement the RAN and Core for everyone to see, mobile network operators usually buy proprietary implementations of all the systems from the same vendor. This is because the different entities that form a network are sold as "black boxes" and mixing and matching between sellers results in an inefficient system with lower performance.

This opaqueness helps equipment vendors to protect their valuable algorithms, considered a business' intellectual property, but it enforces a vendor lock-in that lacks transparency and efficiency. Therefore there is significant opportunity in an open-source implementation that would benefit from the interoperability and flexibility to choose different vendors for a network deployment.

By striving to address these issues, the Open RAN concept is created as an open-source RAN alternative to deal with the increase in data traffic, mobile networks and users in a flexible way; putting its focus on virtualized, intelligent, energy efficient SDR solutions.

2.2.1 O-RAN

To develop this concept, in 2018 the industry lead O-RAN Alliance community of operators, vendors and academic institutions was founded. O-RAN Alliance's mission is "to re-shape the RAN industry towards more intelligent, open, virtualized and fully inter-operable mobile networks." [9]

O-RAN is commonly referred to as non-3GPP, which simply means that a technology has not been standardized by the 3G Partnership Project, and another institution is the one defining the standard. A well-known example of a non-3GPP technology is WIFI, based on the 802.11 family of standards by the IEEE.

The O-RAN project is based on two main pillars: Openness and Intelligence. The former pursues service agility and cloud scale economics in the RAN by creating open interfaces to allow small vendors and operators to introduce services and custom features to the network. This will enable interoperability in the deployments and faster hardware and software development. The latter predicts that, as networks become more complex, dense and demanding, less human intervention is going to be possible. Instead, networks should be automated for self-deployment, optimization and operation to reduce expenses. Therefore, AI/ML techniques have to be embedded in the RAN architecture to provide intelligence. [10]

In line with its principles, O-RAN will focus on software defined, AI enabled RAN Intelligent Controller, RAN virtualization, open interfaces, white box hardware and open-source software.

2.2.1.1 O-RAN Architecture

The O-RAN architecture found in Fig. 2.10, is built around the 3GPP 7.2 RAN split with is focus on intelligence and openness to become the reference for next generation RAN, by designing the virtualized RAN on open-source hardware, with AI controllers. To achieve it, O-RAN defines a series of functional modules such as network management and orchestration, RAN split layers (CU, DU, RRU) and internal open interfaces. The main features of the reference architecture include the RAN Intelligent Controller that the specifications divide by latency demands in non-Real Time

(RT) RIC for >1s control and in near-RT RIC for less than 1s functions.

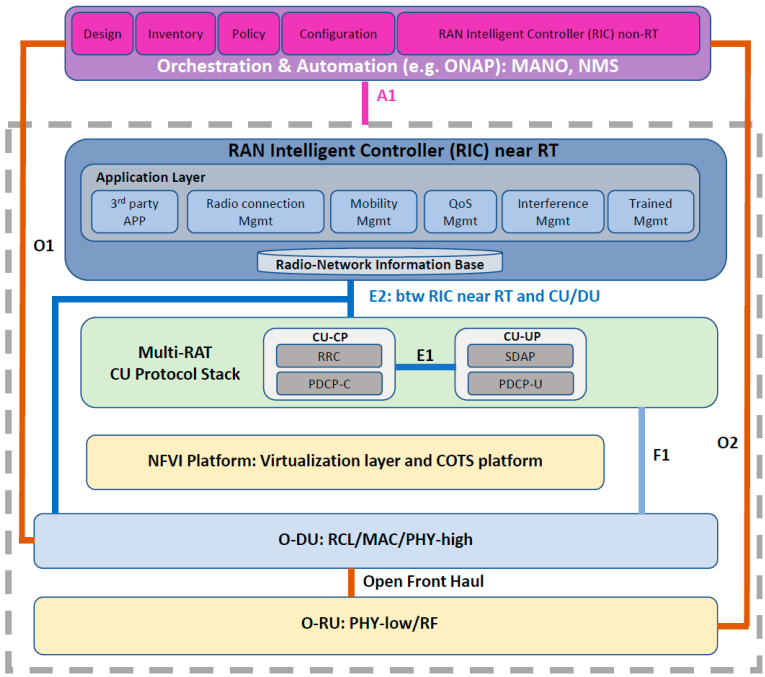


Figure 2.10: O-RAN Architecture. [10]

In the highest layer of this architecture lays the Service & Management Orchestration platform, where the non-RT RIC is located. It includes functions for service and policy management, RAN analytics and ML models training to be distributed to the near-RT RIC for runtime execution. The A1 interface connects it to the gNB (gray dotted box) where the near-RT RIC is found. This interface allows for the network management applications to receive data from the gNB and to act with the AI models sent to the near-RT RIC.

In between the A1 interface and the gNB lower layers is located the near-RT RIC, a logical module that controls and optimizes radio resources with very low latency requirements. It can host 3er party apps called xApps to introduce new functionality to the controller. The E2 interface allows the exchange of data between the xApps and the gNB.

In the lowest layers of this architecture, the split gNB layers are found. The interfaces are mainly provided by 3GPP but enhanced to support interoperability.

2.2.1.2 O-RAN E2 Interface

The O-RAN Alliance effort to create standardized RAN controller interfaces has resulted in the E2 interface that FlexRIC adopts in order to remain compatible. This interface is the responsible for interconnecting RAN elements (called “E2 nodes”) like a gNB, and a near-RT RIC.

The E2 interfaces’ objectives are (i) exposure of selected E2 Node data such as statistics or configurations and (ii) enabling the Near-RT RIC to control selected functions of the RAN like triggering procedures or installing policies. [11]

A Near-RT RIC (henceforth just RIC) provides a terminating point for the E2 interface and controller-internal applications (iApps). It can host external applications (xApps) and additional services including database functionality. A E2 Node is composed of a E2 Agent and several RAN functions, controllable functionalities of the RAN to manage handovers for example. A single RIC is designed to allow for multiple E2 Nodes to be managed through the E2 interface, connecting xApps to RAN functions, as seen in Fig. 2.11

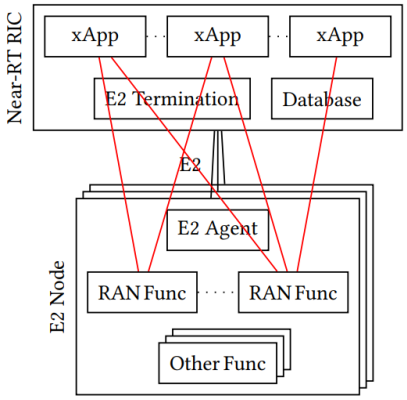


Figure 2.11: The E2 interface allows RIC xApps to control RAN functions. [11]

The E2 interface handles two planes of interaction. One between the RIC and E2 nodes through the E2 application protocol (E2AP) and another between the xApps and RAN functions enabled by E2 service models (E2SM).

E2AP The Application Protocol for E2 management and SM encapsulation divides in two message classes:

- E2 Global Procedures. Messages to manage the connection. (e.g. `E2 Setup Request`)
- E2 Functional Procedures. Messages with destination to the RAN functions that can be divided into:
  - Subscription. xApps can subscribe to event triggers in RAN functions for them to perform an action when this happens. (e.g. `RIC Subscription Request`)
  - Indication. Allow RAN functions to send information to the RIC (e.g. `RIC Indication`)
  - Control. For xApps to execute a procedure at a RAN function.

E2SM E2 Service Models (E2SM) expose the information of a specific RAN function towards the RIC over the E2 interface, embedded within a E2AP message as raw bytes. The E2AP is composed of four services which, combined in different ways, implement an E2SM:

- Reports: A form of data container with pre-defined information sent from RAN functions to xApps on pre-defined trigger events. An xApp would first subscribe (E2AP subscription) to receive *reports* when an event is triggered, and once the RAN function detects the event, the report (E2SM report) is sent over a E2AP indication message.

- Inserts: Messages to inform the xApp that an event that it has subscribed to has been triggered. These messages are transported over a E2AP indication message.
- Control: Messages used by the xApps to order the RAN functions to execute an operation, transported through a E2AP control message.
- Policies: used by the xApp to notify the RAN function to perform predefined operations upon a trigger event. These policies are informed with E2AP subscription messages.

The O-RAN Alliance Work Group 3 has defined three service models [12]:

- E2SM KPM: designed to report key performance metrics from the RAN. During the E2 setup, the gNB specifies the metrics it can report so that an xApp can subscribe to the KPM it is interested in. The gNB will send E2 Indication messages making use of the report service.
- E2SM NI: it allows to manage network interfaces by forwarding messages received from the gNB on a specific interface and reporting them to the RIC via the E2 interface.
- E2SM RC: its goal is to control the RAN for radio resource management optimization in the gNB.

2.3 Open Air Interface

Open Air Interface is an open-source software platform to develop software defined mobile communications networks, based on the 3GPP standards. It is managed by Eurecom, a French telecommunications research institute. The OpenAirInterface is divided in two software development projects: Openair-cn, for the core network; and openairinterface5g, for the RAN. It has implemented solutions for 4G LTE and 5G NR standards. This software-based networks reduce implementation and deployment cost, increase the flexibility to update a deployment and allow fast-prototyping to test its solutions in a lab environment.

2.3.1 FlexRIC

The main Open RAN design is a non-3GPP standard defined by the O-RAN Alliance, who also has a public open-source implementation of their standard under the O-RAN Software Community (OSC) association. Other companies and groups have also showed interest in implementing their own Open RAN compatible modules. One of these organizations is the OpenAirInterface Software Alliance, whose founding member is Eurecom, and who develops the OAI 5G RAN and 5G CN.

Through their project group MOSAIC5G, they aim to transform the RAN and CN into agile and open network-service delivery platforms. This group is providing their Open RAN implementations of the O-RAN E2 protocol named E2 Agent and a flexible RAN Intelligent Controller named FlexRIC [13]. As this is the software implementation Open RAN by the same organization who also develops the OAI 5G RAN software, it is of high interest to study how it works.

The first public release of MOSAIC5G's FlexRIC and E2 Agent was announced at the *"Fall 2021 OpenAirInterface Workshop: Hands-On with the OAI Architects"* event in December 2021 [14].

At the workshop, MOSAIC5G introduced their implementation and showcased it through some demos, as well as proposing several challenges regarding SM and xApps.

The first public release of the code was in late-January 2022 through Eurecom's Gitlab project FlexRIC, as the `master` branch [15]. FlexRIC can be described as a flexible RAN intelligent controller that is composed of a RAN agent and a real-time controller, compliant with O-RAN E2AP specification.

MOSAIC5G's project group workflow has not been to publicly develop FlexRIC through open commits in Gitlab, but to work privately and develop different Proofs-of-Concept and later release it all at once deciding which PoC to keep and cleaning the code. Therefore, there have been very seldom commits since early 2022 until late-July / early-August when the `dev` version has been updated.

In Fig. 2.12 a simplified architecture of the FlexRIC SDK is provided to indicate how the E2 Interface Agent and Server libraries are used to communicate the E2 Node with the RIC. Inside the controller are located the iApps that manage the controller's own services and SMs. iApps can communicate externally to xApps through what FlexRIC designers have called the E42AP Interface. This interface transmits the xApp's requests and subscriptions to RAN Functions, and the SM reports.

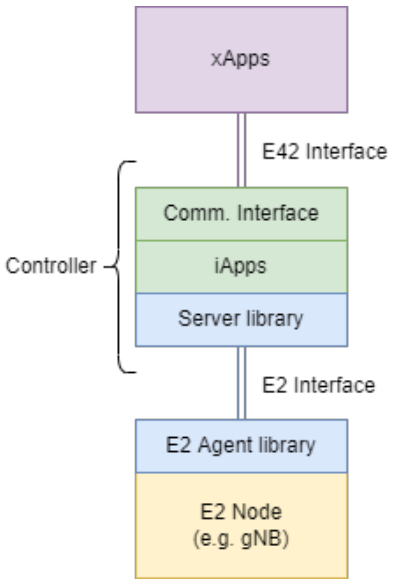


Figure 2.12: FlexRIC Architecture.

In its first version, FlexRIC provides MAC, RLC, and PDCP statistics monitoring service models based on the O-RAN E2SM KPM. This SMs allows the RIC to subscribe and receive relevant statistics from the gNB about the mentioned layers. The full list of FlexRIC's `master` branch E2AP supported stats indicators are available in the Appendix B.

2.4 Universal Software Radio Peripherals

In the recent years a new trend has been established for radio communications networks: Software Defined Radio (SDR), which replaces analog RF hardware components with software programmable components for the radio heads. This SDR devices make use of digital signal processors, general purpose processing and field programmable gate arrays to implement the RF functions needed to transform the digital signals to analog. It allows the radio devices to be re-configurable and lower the time and cost for prototyping network architectures.
The SDRs can be programmed to comply with different technologies; different standards; and vary the carrier, bandwidth, MIMO… configurations. It allows users to transmit and receive many different waveforms at various frequencies and settings on a common hardware platform.

A widespread product family of SDR are Universal Software Radio Peripherals (USRP) by *Ettus Research* and its parent company *National Instruments*. These peripherals connect to a host PC through a serial bus or network connection, and are able to perform the instructions or functions programmed in the host PC via the USRP Hardware Driver (UHD).
The USRP family embraces the open-source philosophy both in hardware and software, making it a common product for research, prototyping and deploying radio communications systems in industry labs and universities.

Ettus Research designs the following product series:

- Bus series (B): Connection through USB 3.0. Most common device is the B210.
- Networked series (N): Connection through Ethernet. Popular devices include the N310 and N320.
- High Performance series (X): Connection can be Ethernet or a x4 PCIe.
- Embedded series (E): Can run in stand-alone mode -without a host computer- with an integrated OS.

The interface to communicate with and control the USRP from a GPP host computer is the UHD library. It is an open-source firmware written in C/C++ that provides the necessary control to transport user waveform samples to and from the USRP as well as control various hardware parameters such as sampling rate, frequency, gains, etc.

This driver includes some utilities and commands such as `benchmark_rate`, a benchmark to test the network link at different bandwidths; `uhd_find_devices`, a command to check the USRPs available in the network; and `uhd_usrp_probe`, a command to list a USRP's parameters.

Chapter 3

Deployment's Setup

In this chapter, the lab setup utilized for the RAN deployment is discussed. It first focuses on the PC and USRP hardware and its configuration, followed by the FlexRIC and OAI software install.

3.1 Host PCs

OAI is designed to be run in a machine and, as the plan is to use two USRPs for UE and gNB, on PC will be used for each. The new 5G implementations in OAI are very resource-intensive mainly because of the LDPC encoding and decoding. If we also add the USRP's processing, then the result is that very powerful machines need to be used.

The gNB has been deployed on a PC with an Intel i7-4790 CPU @ 3.60GHz and 16GB of RAM memory. The UE has been deployed on a PC with an Intel i5-4460 CPU @ 3.20GHz and 16GB of RAM memory. Both systems have Ubuntu 18.04 LTS OS installed.

Apart from using relatively powerful machines, some configuration parameters need to be tweaked to get the most out of them.

3.1.1 Low latency Kernel

As the computations that need to be performed are in real-time with very low tolerance to delays, a Linux low-latency Kernel needs to be installed in the following way:

```
$ sudo apt-get install linux-image-lowlatency linux-headers-lowlatency
    - Reboot and select the low latency kernel in the GRUB menu, under
      Advanced options for Ubuntu. -
$ uname -r      - Once booted, check the kernel. You should expect
   something similar to 4.15.0-109-lowlatency -
```

3.1.2 CPU power management

The CPUs previously mentioned have 4 cores and 8 threads. When using Hyper-Threading, the Operating System (OS) shows each thread as a core and, although it enables more overall power, each core is less powerful itself. Then working in multi-threading environments this is helpful but OAI does not employ much threads and thus it is more important to have fewer, more powerful ones. To do so, in the BIOS settings we must disable Hyper-Threading to only have the 4 true cores.

In the BIOS settings we should also disable any power management functions such as the C-states (Sleep states) and CPU frequency scaling (Intel SpeedStep). This configuration has to be checked and updated in Linux as well.

```
$ watch grep \"cpu MHz\" /proc/cpuinfo      - To make sure the CPU
    frequency is always at its maximum -

$ sudo nano /etc/default/grub
    - Change RUB_CMDLINE_LINUX_DEFAULT="quiet intel_pstate=disable
      processor.max_cstate=1 intel_idle.max_cstate=0 idle=poll" -
$ sudo update-grub

$ sudo nano /etc/modprobe.d/blacklist.conf
    - Paste blacklist intel_powerclamp -

$ sudo i7z      - To check that the C-state 0 is the only one with 100%
    and all others are at 0% -
```

3.1.3 CPU governor

We must ensure that the CPU governor is set to performance to set the CPU frequency to its highest within the borders of its scaling limits.

```
$ sudo apt-get install cpufrequtils
$ sudo vi /etc/default/cpufrequtils    - Add GOVERNOR="performance" -

$ sudo update-rc.d ondemand disable    - To make this change permanent
    -
$ sudo /etc/init.d/cpufrequtils restart
$ cpufreq-info    - All CPU cores should be in performance mode -
```

3.1.4 KPTI Protections

Linux's systems provide KPTI protections against Spectre and Meltdown hardware vulnerabilities. This protections can decrease performance by up to 15%. For systems that require the maximum performance and after understanding the ramification that this modification can have from a security standpoint, KPTI protections can be disabled [16].

```
$ sudo nano /etc/default/grub
    - Add GRUB_CMDLINE_LINUX_DEFAULT="pti=off spectre_v2=off l1tf=off
      nospec_store_bypass_disable no_stf_barrier" -
$ sudo update-grub
```

3.2 USRP N310

In the iTEAM's MCG there have been several works on the use of USRP B210 and recently N310s were acquired, as it is a newer and more capable product. There hasn't been much research work done in the group on networked USRPs and they present some specific challenges, especially when using large bandwidths.

This project is deployed on two N310 USRPs with the following key features: [17]

- 4 RX, 4 TX channels
- 10 MHz – 6 GHz extended frequency range
- Up to 100 MHz of instantaneous bandwidth per channel
- Two SFP+ ports (1 GbE, 10 GbE, Aurora)
- One RJ45 (1GbE)

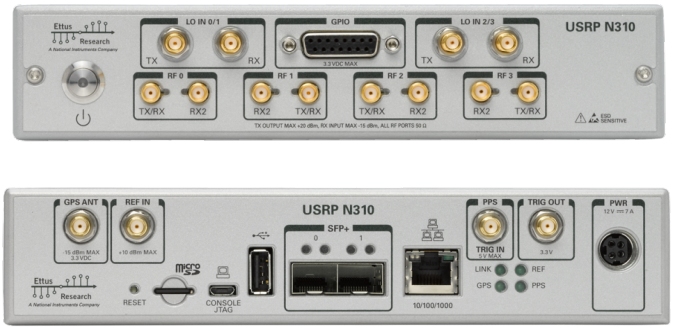


Figure 3.1: USRP N310 front and back panels. [17]

Both PCs are connected to two Ettus USRP N310 as the radio heads. The link between the PCs and the USRPs is a SPF+ to SPF+ 10Gbps cable that connects a 10Gbps capable SFP+ port on the USRP (port 1) to a 10Gigabit Ethernet PCIe Card on the PC. The SFP+ cable from Ettus has part number 783344-01 and the PCIe card has part number 783345-01.

3.2.1 Sampling rate

The USRP N310 supports three Master Clock Rates: 122.88Mhz, 125MHz and 153.6 MHz. Only strict integer decimation and interpolation are used within USRP hardware to achieve the desired

sample rate, and OAI is only compatible with some ratios based on a 122.88MHz Master Clock Rate. [18]

Master Clock Rate	Decimation / Interpolation Rate Host Sample Rate [Samples per second]				
1	2	4	6	8	10
122.88e6	61.44e6	30.72e6	20.48e6	15.36e6	12.288e6
125e6	62.5e6	31.25e6	20.833e6	15.625e6	12.5e6
153.6e6	76.8e6	38.4e6	25.6e6	19.2e6	15.36e6

Table 3.1: USRP N310 common sampling rates. [18]

Also, OAI's standard 5G-NR configuration is on frequency band 78 and 106 Physical Resource Blocks (PRB). The bandwidth is then calculated with 12 subcarriers spaced 30Khz in each of the 106 PRBs which results in 38.16 Mhz, commonly referred to as a 40Mhz bandwidth.

Moreover, as the signal processing is done in band base, the sample rate must be at least the same as the bandwidth to comply with Nyquist Theorem.

Therefore, the lowest sample rate that meets these requirements is 61.44MSps in both ways for Full Duplex mode, or 122.88MSps total over the link. As the maximum sample rate over a 1 Gigabit link is 25MSps and 200MSps for a 10 Gigabit link, we concluded that a 10Gbps link is required between the PCs and USRPs for this bandwidth.

3.2.2 Network configuration

As stated previously, the link between the PCs and USRPs is a 10Gigabit SFP+ connection. By default, with the *HG* FPGA image loaded, only SFP Port 1 will support 10Gb speeds and it is assigned IP address 192.168.20.2. The SFP Port 0 and the RJ45 Management Port are 1Gb and the former has IP address 192.168.10.2 while the latter is configured for DHCP.

It is possible to view and change the USRP network settings by accessing it via SSH through the RJ45 Management Port, SFP Ports or using the serial connection and changing the configuration stored in /data/network/[eth0 - sfp0 - sfp1].network.

```
eth0 (DHCP):

  [Match]
  Name=eth0

  [Network]
  DHCP=ipv4

  [DHCP]
  UseHostname=false
```

```
sfp0 (static):

  [Match]
  Name=sfp0

  [Network]
  Address=192.168.10.2/24

  [Link]
  MTUBytes=1500
```

```
sfp1 (static):

  [Match]
  Name=sfp1

  [Network]
  Address=192.168.20.2/24

  [Link]
  MTUBytes=9000
```

Ettus' documentation on the N310 USRPs states that MTU should be 9000 for 10Gb streaming speeds and 1500 for 1Gb [18]. For some unknown reason, the USRPs came with MTU 8000

which had to be changed to 9000. On the Host PC, the network interface settings must be changed accordingly:

```
IP Address: 192.168.20.1
Subnet Mask: 255.255.255.0
Gateway: 0.0.0.0
MTU: 9000
```

The communication through this link uses UDP as the transport protocol. On Linux, the maximum buffer sizes must be changed depending on the streaming speed to prevent dropping packets due to lack of CPU computation power, which generates D (dropped frames) errors. These errors can be quite frequent, appearing at random intervals and usually in single events, and are catastrophic when using OAI. It can be quite difficult to get rid of this error and, although *Ettus* recommends some possible tuning guidelines, there are other that can be useful to [19].

Linux's kernel distinguishes between TCP buffers and all protocol (core) buffer, so to increase the UDP buffer we must change the core read and write maximum memory values. The receive buffer cannot overflow because the peer is not allowed to send data beyond the buffer size window. This means that datagrams will be discarded if they do not fit in the socket receive buffer. 62.5MB is the minimum required by UHD at this speed, and it displays a warning if it is lower, but increasing it might be helpful to decrease the frequency of D errors in systems that lack CPU power. If the CPU cannot keep up with the data stream, errors will be inevitable, although a large buffer makes it possible to have longer runs without dropping frames. To change the maximum buffer sizes, you run:

```
$ sudo sysctl -w net.core.rmem_max=62500000
$ sudo sysctl -w net.core.wmem_max=62500000
 - or permanently change the buffer value -
$ echo 'net.core.wmem_max=62500000' >> /etc/sysctl.conf
$ echo 'net.core.rmem_max=62500000' >> /etc/sysctl.conf
```

You can also change the Network Interface Card's (NIC) RX and TX ring buffer queues . First you can check the current and maximum settings and then change them to the `Pre-set maximums` values by running:

```
$ ethtool -g <interface>
$ sudo ethtool -G <interface> tx <N> rx <N>
```

In this case, the NICs allowed to change from 512 to 4096 bytes. Be aware that changing the ring buffers can affect the latency by delaying the delivery of datagrams to the UHD.

Finally, if each CPU core is not powerful enough by itself to handle the streaming rate, it is possible to change the Interrupt Request's (IRQ) `smp_affinity` so that, even if only one core can process incoming packets at a time, any interruption causing delay on the incumbent core can be solved by allowing another core to keep up with the stream. You can check the number of interrupts per CPU core and I/O device in real-time while streaming data to/from the USRP with the command:

```
$ watch -n 0.1 cat /proc/interrupts
```

Then check the `smp_affinity` of an IRQ process number. And finally changing the default value to prevent overwhelming a single CPU core, mainly number 1 as this is the one that carries more application workload.

```
$ cat /proc/irq/32/smp_affinity
    1   - IRQ 32 is assigned CPU core 1 -
$ echo 3 >/proc/irq/32/smp_affinity
```

After configuring the USRP and host PC, it is possible to benchmark the system using the UHD utility `benchmark_rate`. To test the link requirements, we use the following command, which will generate a full-duplex stream of 2Gbps to and from the USRP for 60 seconds.

```
$ sudo ./usr/local/lib/uhd/examples/benchmark_rate  \
   --args "addr=192.168.10.2,master_clock_rate=122.88e6" \
   --duration 60  --rx_rate 61.44e6  --tx_rate 61.44e6
```

3.2.3 Link between the USRPs

To take any over-the-air problems out of the equation, it is recommended to first try running OAI without the air channel so the USRPs will be connected directly by cable in loopback mode.

As we are using daughterboard 0, channel 0 by default, there must be one SMA to SMA cable running from the `TX/RX_RF0` port on one USRP to the `RX2_RF0` port on the other with a minimum attenuation of 30dB between them. In this case, two 20dB attenuators have been added to each cable for a total of 40dB. The maximum input power is +15dBm to prevent damaging the daughterboards, and it can be checked using an RF Spectrum power meter.

3.2.4 Time and clock synchronization

Although OAI does not explicitly require to synchronize the RF boards, 5G base stations usually contain GPS modules to sync and establish a common time scale. To improve the OAI performance and reduce the possibility of divergences, I have opted to sync both USRPs. USRP N310 have an internal Pulse Per Second (PPS) time and 10Mhz clock reference that is used by default. The PPS signal is shown in PPS LED in the back panel. It is obvious that without external synchronization the PPS LED of each USRP will pulse at different times. The clock reference is also expected to diverge between them, and although there is an option in OAI to compensate for this frequency offset (–ue-fo-compensation), it is best to sync them in hardware.

As external time and clock reference two options are available: a GPS Disciplined Oscilator (GPSDO) or an external input. The Lab where the USRP testbench is located is an indoor enviroment where GPS signal is very weak, so an external reference in the form of an *Ettus' OctoClock-G CDA-2990* will be used. [20]

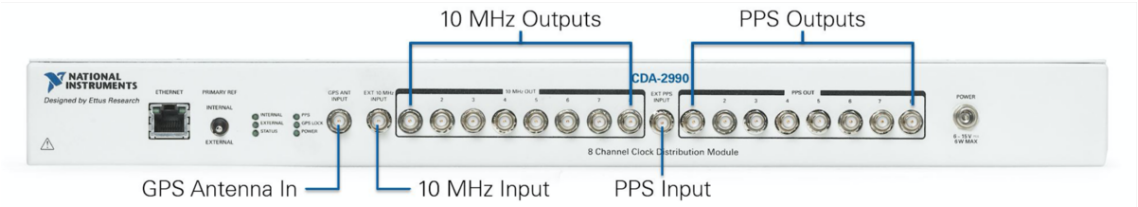


Figure 3.2: OctoClock-G CDA-2990 front panel. [20]

The *OctoClock* can distribute a 10Mhz and PPS signal across 8 devices trough SMA cable connections, without the need for a GPS signal (although it can be used with it). Its set up consists of connecting the 10Mhz outputs to the `REF IN` input and the PPS outputs to the `PPS/TRIG IN` input of the USRPs, and powering on the *OctoClock*. Then indicating the UHD to select the external reference with the `--args` option, which will reboot the daughterboards to change the reference. Now the PPS LED of the USRPs and *OctoClock* will blink in sync.

```
--args "addr=192.168.10.2, master_clock_rate=122.88e6, time_source
   =1, clock_source=1"
```

Finally, the Lab setup is as shown on the diagram in Fig. 3.3 and on the photo in Fig. 3.4.

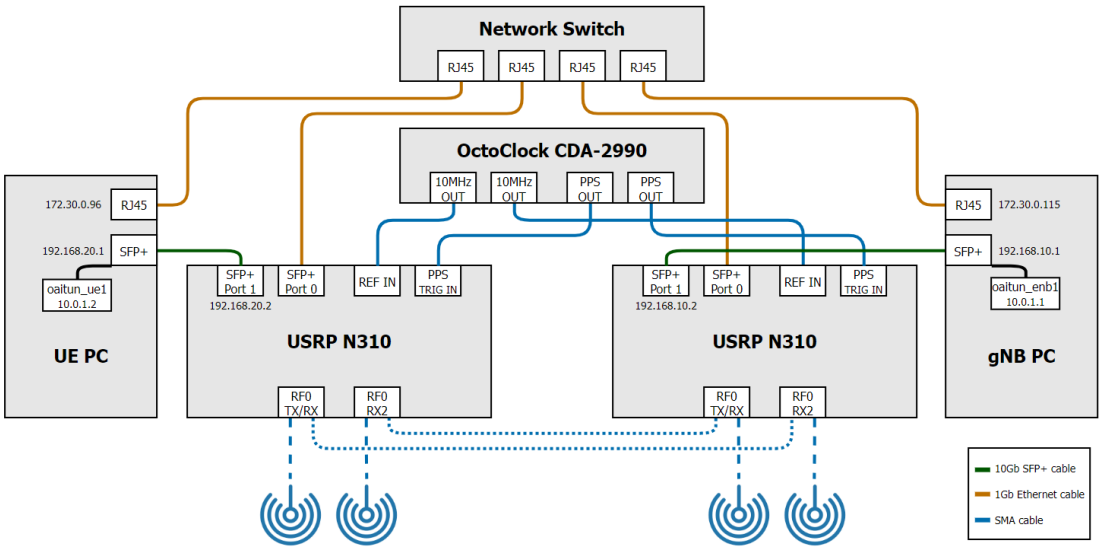


Figure 3.3: Deployment setup diagram.

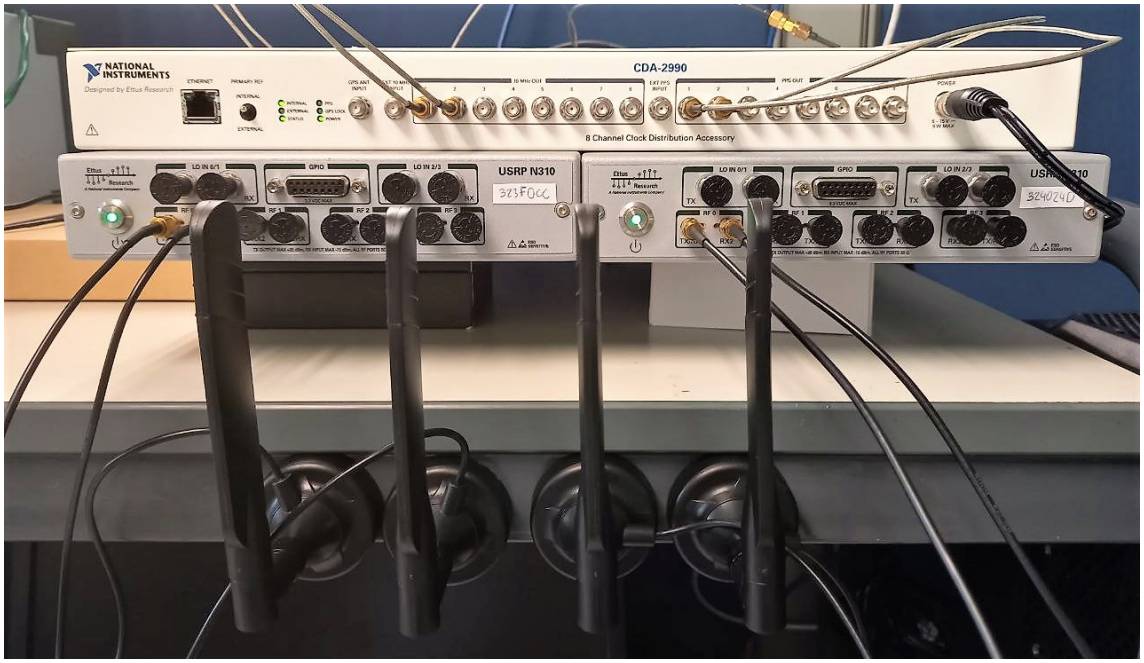


Figure 3.4: Deployment setup in the Lab.

3.3 FlexRIC & OAI Deployment

FlexRIC's master branch, released early 2022, provides a E2 Agent and near-RIC. The E2 Agent can be integrated with a single Radio Access Technology (RAT), OAI 5G RAN, so that the gNB can communicate with the E2 node at the near-RIC. This integration is done with a git patch which modifies some OAI files to create the E2 agent and E2AP messages at the gNB.

This master branch is integrated with OAI commit `b04731d7565cd91b538eb7cc80f874b4730d54ad (integration_2021_wk51_a)`, that defines the tag `2021.w51_a` from December 2021.

3.3.1 FlexRIC install

The FlexRIC's Gitlab repository indicates the following instructions to build and install the master branch on the machine where we plan to run the gNB from. As we plan to run the E2 Agent and the Near-RIC on the same machine, we will only have to install it once. It is possible to run them in different machines changing the Near-RIC IP address at `/usr/lib/FlexRIC/FlexRIC.conf`.

To build the repository on Ubuntu, they recommend installing `CMake` through a Personal Package Archive (PPA) such as `Kitware APT Repository`. To do so, `Kitware` provides a shell script.

```
$ vim kitware-archive.sh          - Create the file -
          - Paste the script and save with :wq -
$ chmod +x kitware-archive.sh     - Make the file executable -
$ sudo ./kitware-archive.sh       - Run the script -
$ sudo apt-get install cmake      - Install CMake -
          - Install the required dependencies -
$ sudo apt install libsctp-dev poppler-utils python3.8 libreoffice
```

The asn1c package available in apt is v0.9.28 but the v0.9.29 is needed, so it must be installed from source from its github repository. Asn1c requires some dependences (automake-1.15, libtool, bison=2.x (2.7.1), and flex) that must be installed as well with its correct version.

```
    - Install dependencies -
$ sudo apt-get install automake libtool flex
$ wget http://launchpadlibrarian.net/140087283/libbison-dev_2.7.1.dfsg
   -1_amd64.deb
$ wget http://launchpadlibrarian.net/140087282/bison_2.7.1.dfsg-1_amd64
   .deb
$ sudo dpkg -i libbison-dev_2.7.1.dfsg-1_amd64.deb
$ sudo dpkg -i bison_2.7.1.dfsg-1_amd64.deb
$ sudo apt-mark hold libbison-dev sudo apt-mark hold bison

    - Install asn1c from source -
$ git clone https://github.com/vlm/asn1c
$ test -f configure || autoreconf -iv
$ ./configure
$ make
$ sudo make install
```

The xApp communication is done with Next Generation of Nanomessages (nng), that must be installed from its repository.

```
$ git clone https://github.com/nanomsg/nng
$ cd nng - && mkdir build && cd build -

    - Install ninja to build the files as recommended -
$ sudo apt-get install ninja-build

$ cmake -G Ninja ..
$ ninja
$ ninja test
$ sudo ninja install
```

Now we can clone the FlexRIC repository into a folder named `FlexRIC` inside our working directory.

```
$ git clone https://gitlab.eurecom.fr/MOSAIC5g/FlexRIC.git
$ cd FlexRIC
```

The E2AP messages defined by O-RAN must be acquired directly from [21] after signing the O-RAN ADOPTER LICENSE AGREEMENT. The E2AP messages definitions can be found at the `O-RAN.WG3.E2AP-v01.01.pdf` file.

The MOSAIC5G team provides a python script to convert the ASN.1 module files into C structs. This script takes the O-RAN E2AP definitions file in .docx or .pdf format as an input, converts it to pdf, then extracts only the messages definitions to a text file (`e2ap-v01.01.asn1`) and runs the asn1c command. There is a known problem during this step that generates errors due to the wrong conversion of the pdf to text, as there are some doble spaces characters that asn1c doesn't recognize. To fix this, the txt must be edited by hand to remove the doble spaces and then run the asn1c command from the console.

```
    - The script can be found here: -
$ cd src/lib/ap/ie/asn
$ python3 gen_asn.py path/to/O-RAN.WG3.E2AP-v01.01.docx or .pdf

    - The asn1c command inside the script is: -
$ asn1c -gen-PER -no-gen-OER -fcompound-names -no-gen-example -
   findirect-choice -fno-include-deps e2ap-v01.01.asn1
```

Once the C structs are created at the `src/lib/ap/ie/asn` path, we can go back and compile the project.

```
$ cd ../../../../../ && mkdir build && cd build && cmake ..
$ make -j
$ sudo make install
```

To check if the project has been correctly installed, we can run a E2-Agent Near-RIC test that will simulate MAC, RLC and PDCP communication through the E2 interface and store it in a log.txt file.

```
$ sudo ./FlexRIC/build/test/test_near_ric
```

The Standalone Near-RIC is started by running `near_ric_sa`, and stopped by pressing `CTRL+C`.

```
$ sudo ./FlexRIC/build/test/near_ric_sa
```

3.3.2 OAI install and integration

For the E2-Agent to be included inside the OAI gNB, a patch must be applied to the OAI repository [22], in a process that's beeing called *integration*.

First we clone the repository and select the commit that corresponds to the tag `2021.w51_a`. Then the git-am command applies a series of patches from a mailbox. Git might ask for a username and email to sign the changes to the repository, indicating the commands to follow.

```
$ git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git oai
$ cd oai && git checkout b04731d7565cd91b538eb7cc80f874b4730d54ad
$ git am ../FlexRIC/oai/FlexRIC_oai.patch    - Applying the patch -
```

Once the patch is applied, we can proceed building OAI as usual with the `./build_oai` executable. At the gNB machine the `-gNB` option will build the `./nr-softmodem` executable, while option `-w` USRP indicates to build the libraries to work with UHD. Option `-I` installs the requirements. To guarantee that UHD version is the same as the USRP image (v3.15), we have to indicate it.

```
$ export BUILD_UHD_FROM_SOURCE=True
$ export UHD_VERSION=3.15.0.0
$ cd cmake_targets && ./build_oai -c -C -I -w USRP -gNB
```

At the UE machine there is no need to install FlexRIC (as the UE is agnostic to the O-RAN architecture) and only OAI has to be installed, indicating we want to build the `./nr-uesoftmodem` executable with option `--nrUE`.

```
$ git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git oai
$ cd oai && git checkout b04731d7565cd91b538eb7cc80f874b4730d54ad
$ export BUILD_UHD_FROM_SOURCE=True
$ export UHD_VERSION=3.15.0.0
$ cd cmake_targets && ./build_oai -c -C -I -w USRP -nrUE
```

This would conclude the installation, with OAI ready to be executed as the next chapter explains.

Chapter 4

Methodology

In the development of this project, I make use of different OAI modes of operation and options, which are described in this chapter, along with the software and hardware tools utilized, in order to analyze the deployment and debug the errors encountered. A brief description of FlexRIC and its xApps and how to execute them is also explained. Finally, the FlexRIC xApp challenge is introduced. This explanation will help understand how the results are accomplished.

4.1 OAI RAN

OpenAirInterfece has available several modes in which to run its softmodems. By default it operates in Non Stand-Alone mode, but it is possible to change it to Standalone with the `-sa` option. As the scope of this project is to analyze the MOSAIC5G implementation of the OpenRAN architecture, no Core Network has been deployed, only the OAI 5G RAN. As a result of this, the UE cannot establish a PDU Session with the AMF and, although it can decode the PBCH and SIB1, it cannot attach fully to the gNB

Nonetheless, the SA mode has been tested to inspect the available random access procedure and confirm that no attachment is possible.

4.1.1 phy-test & noS1

For cases where only the RAN is deployed, OAI has introduced the `phy-test` mode to connect a OAI UE to a OAI gNB without the support of a CN. In this mode the gNB will do the following:

- It reads the RRC configuration from the config file.
- It encodes the RRCConfiguration and the RBconfig message and stores them in the binary files rbconfig.raw and reconfig.raw.
- The MAC layer uses a pre-configured allocation of PDSCH and PUSCH with randomly generated payload and schedules continuous transmission even without a UE.

At the UE this mode will:

- Read the binary files rbconfig.raw and reconfig.raw and process them.
- Decode the SIB1 message from the rbconfig.raw file.
- Use a pre-configured schedule to transmit to the gNB.

Summarizing, in this mode the gNB's RRC configuration is not obtained via the usual attachment procedure but by reading some files, and random UL and DL traffic is generated at every scheduling opportunity. For the `phy-test` mode to work, the rbconfig.raw and reconfig.raw files generated by the gNB with the desired configuration file has to be copied to the UE machine at the same directory.

The phy-test mode allows for some options to configure the scheduler on the command line:

- `-m` Set the downlink MCS for PHYTEST mode.
- `-t` Set the uplink MCS for PHYTEST mode.
- `-l` Set the downlink nrOfLayers for PHYTEST mode.
- `-L` Set the uplink nrOfLayers for PHYTEST mode.
- `-M` Set the number of PRBs used for DLSCH in PHYTEST mode.
- `-T` Set the number of PRBs used for ULSCH in PHYTEST mode.
- `-D` Bitmap for DLSCH slots (slot 0 starts at LSB).
- `-U` Bitmap for ULSCH slots (slot 0 starts at LSB).

In this mode, instead of randomly generated payload, we can also transmit and receive user-plane traffic over a Linux TUN interface with the `noS1` mode. The S1 interface in LTE is used to communicate the RAN nodes and EPC. Although it doesn't exist in 5G NR (except in NSA mode to connect to the EPC) and it is replaced with the NG interface, this mode has kept its original name from when it was designed in LTE.

TUN interfaces are virtual network devices to create a tunnel to carry IP packets in OSI layer 3 (Network layer). It is used as a network access to route application layer data through the OAI RAN link. This way we can use tools like Ping or Iperf to obtain useful statistics about the network performance.

4.1.2 gNB config file

In OAI, to establish the gNB parameters such as frequency, duplexing mode, bandwidth,... a configuration file is used. OAI provides many for many different scenarios, one of which has been modified to be used with FlexRIC and can be found at Appendix A.

At the gNB we need to pass the config file with option `-O`. At the UE we must indicate the USRP parameters with option `-usrp-args` (equivalent to the `sdr_addr` field in the gNB config file) as well as the Tx and Rx gains to set the USRP to with options `-ue-txgain` and `-ue-rxgain`. With all of this in mind, the commands used to run the gNB and nrUE are:

```
\textit{~/gNB/MOSAIC5G/oai/cmake_targets/ran_build/build} $ sudo ./nr-
    softmodem -O ../../../../FlexRIC/oai /gnb.band78.sa.fr1.106PRB.
    usrpn310.conf -phy-test --noS1

\textit{~/UE/MOSAIC5G/oai/cmake_targets/ran_build/build} $ sudo ./nr-
    uesoftmodem --usrp-args "addr"=192.168.10.2 --ue-txgain 50 --ue-
    rxgain 40 --ue-fo-compensation - phy-test --noS1 --nokrnmod -O
    ../../../targets/PROJECTS/GENERIC-NR-5GC/CONF/ue.conf  --time-
    source -clock-source
```

4.1.3 RF Simulator

RF boards such as USRPs and resource-intensive LDPC algorithms require high performance PC hardware to run, limiting the RAN transmission capabilities and making the system more prone to errors. Instead, OpenAirInterface offers an RF simulator to replace the actual RF boards. The RF Simulator (RFSIM) mimics the RF encoding and decoding as much as possible, but not in real-time. It can run faster or slower than real-time depending on the CPU available resources, making it CPU-bound instead of real-time RF sampling-bound. It can be run in `noS1` and/or `phy-test` modes.

To build this RFSIM as an OAI device, we can build it from scratch with the gNB or UE running the `./build_oai` executable using the option `-w SIMU`, or build it after a regular build by running the following on both machines:

```
$ cd /oai/cmake_targets/ran_build/build
$ make rfsimulator
```

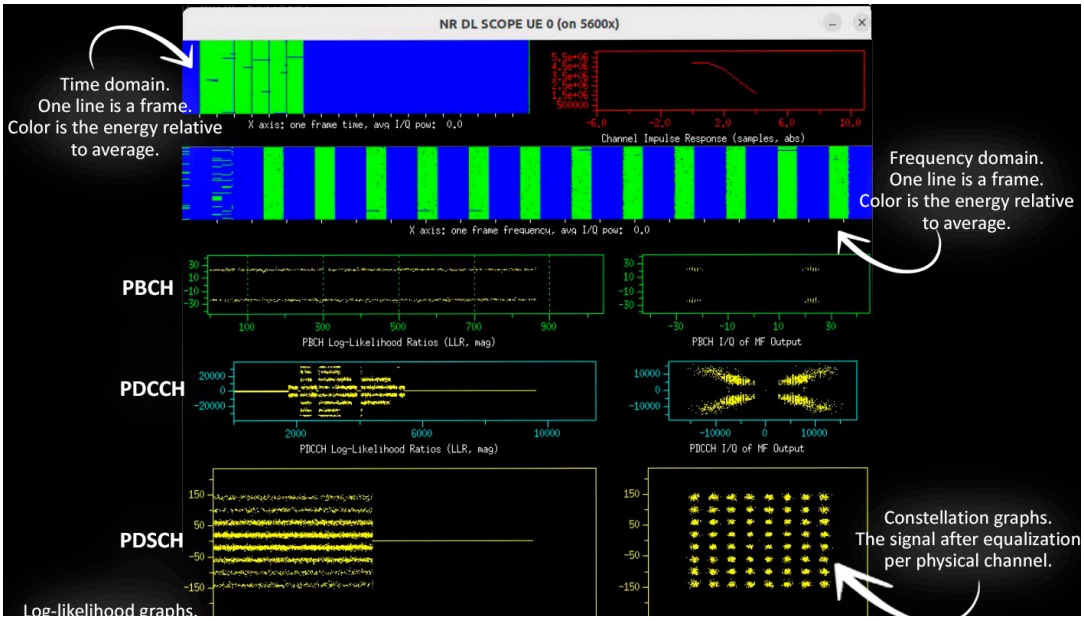
To use the RF simulator, both softmodems have to be run adding the `-rfsim` option to the command line. And to indicate where each machine is located so that they can communicate with each other, the env variable `RFSIMULATOR` can be used to set the gNB node as the simulator server and to tell the UE what IP does the gNB node have. By default it uses port 4043.

```
~gNB  $ sudo RFSIMULATOR=server ./nr-softmodem [...] --rfsim
~UE   $ sudo RFSIMULATOR=172.6.30.115 ./nr-uesoftmodem [...] -rfsim
```

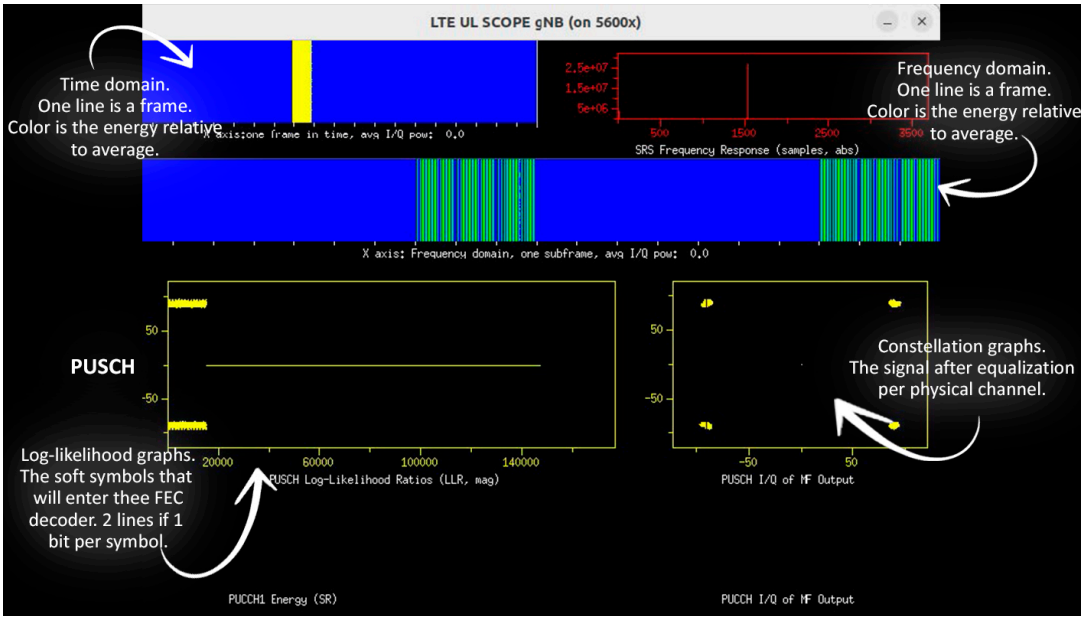
When using the RF Simulator it is possible to simulate custom channel conditions with the option `chanmod`. The channel models can be configured in a separate `channelmod_rfsimu.conf` file linked to the main config file using `@include "channelmod_rfsimu.conf"`. In this file the channel parameters are defined. Some of these parameters can be modified in execution time through the telnet server.

4.1.4 Scope

As part of the debugging tools that OAI has to offer there is a physical layer oscilloscope emulator called Scope, which consists of a graphical interface to display transport channels for both gNB and nrUE. To build the shared libs you only need to run `make nrscope`. Then, with option -d on command line we can run the scope on the gNB and/or the UE. The NR DL Scope (UE) and NR UL Scope (gNB) show the following information.



(a) NR DL Scope



(b) NR UL Scope

Figure 4.1: OAI Scopes explained

As many of the tools in OAI, the scope was developed for LTE and some of the graphs have not been updated to NR such as `PUCCH Energy, PUSCH Throughput, PUCCH I/Q or PDSCH Throughput`. Some of the window names have not been updated either as seen in `LTE UL SCOPE gNB`. This is a common trend all over the OAI code and documentation.

4.1.5 T-tracer & Wireshark

Another debugging tool in OAI is the T-Tracer, a frame decoder and analyzer GUI. To build it, we just need to go to the `/oai/common/T/tracer` directory and do `make`. It can be run by the different executables available at the same directory: `./enb, ./ue or./gnb`. To enable it at the softmodem we must add the option `-T_stdout 0` to disable output on the terminal and only use the T tracer.

```
$ sudo ./nr-softmodem -O gnb.band78.sa.fr1.106PRB.usrpn310.conf --phy-
   test --noS1 -T_stdout 0
$ sudo ./gnb -d ../T_messages.txt
```

As many of the tools in OAI, T-tracer was developed for LTE and it has not been updated to NR, therefore it doesn't log almost any message. Nonetheless, it is possible to use Wireshark [23] to analyse MAC PDUs for UEs, MIBs, SIBs and random accesses seen by the gNodeB with a T-tracer tool called `macpdu2wireshark`. This executable will send some UDP packets to through the local interface for Wireshark to capture. With it we can see live traffic or record and replay traces.

To live capture:

```
$ ./macpdu2wireshark -d ../T_messages.txt -live
```

To record and replay:

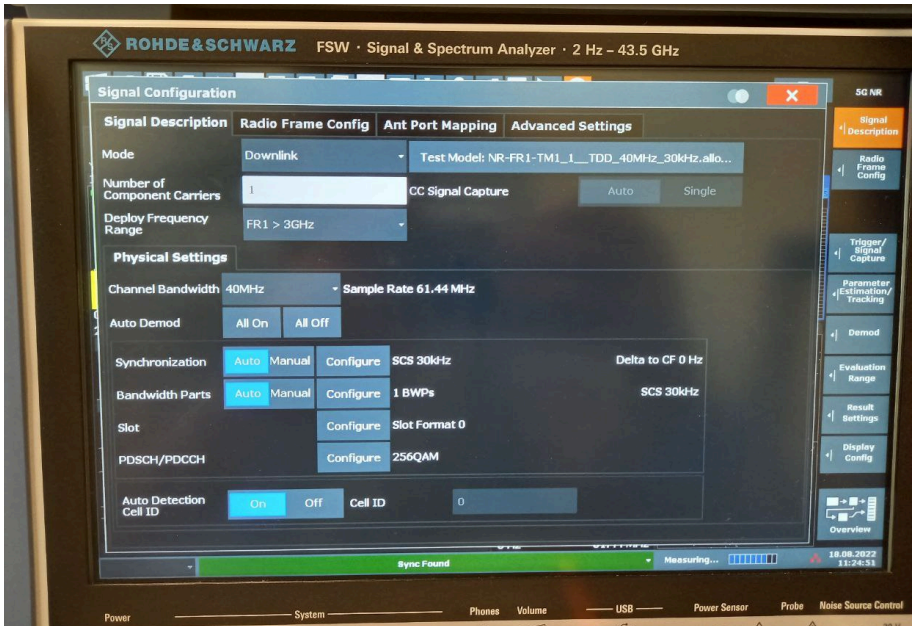
```
$ ./record -d ../T_messages.txt -o /tmp/record.raw -on WIRESHARK
$ ./extract_config -i /tmp/record.raw > extracted_T_messages.txt
    - Open Wireshark and listen to the local interface -
$ ./macpdu2wireshark -d extracted_T_messages.txt -i /tmp/record.raw
```

For Wireshark to dissect the packets, we need to configure it first using a recent version (v3.6.5 in our case) following the instructions provided in OAI's wiki.

4.1.6 Spectrum analyser

The iTEAM's MCG's Lab has a *Rohde & Schwarz FSW Signal & Spectrum Analyser* paired along with the *R&S FSW-K144* measurement application for 5G-NR downlink [24]. With it we are able to visualize the RF signal transmitted from a gNB in downlink. To configure the 5G NR app, the frequency and bandwidth must be selected. As the SSB period defined in the config file is 20ms, the *FSW* signal capture buffer must be changed from the 20ms default to 40ms to find sync. In

the Signal Configuration window the appropriate Test Model can be selected to match the TDD, 40MHz BW, 30kHz SCS, FR1 parameters. The Synchronization, Bandwidth Parts and Cell ID are selected for Auto configuration.



(a) Signal configuration

(b) Signal capture

Figure 4.2: R&S *FSW* configuration.

The *FSW* can also be externally sync with the OctoClock via a 10MHz signal on its `REF INPUT` connector to reduce frequency deviations. To activate the external reference frequency, it can be done in *Setup* > *Reference* > *External Reference 10MHz*.

4.1.7 ROMES

Rohde & Schwarz also makes *ROMES*, a software program that, paired with an adequate hardware decoder and antenna, can scan the spectrum for available networks and provide coverage measurements and key BS parameters [25].

The main difference between a spectrum analyzer and this software is that *ROMES* can automatically scan and decode any available networks for several mobile technologies, providing more in-depth statistics. It can decode the PBCH and obtain the information included in the MIB and SIB1 messages broadcasted by the gNBs. It can also differentiate different cells and beams.

In it we select 5G NR as the technology to decode, the frequency band and start recording. Then in its different windows we select the Automatic Channel Detection (ADC) status, 5G NR Scanner and 5G NR BCH View to view the relevant information.

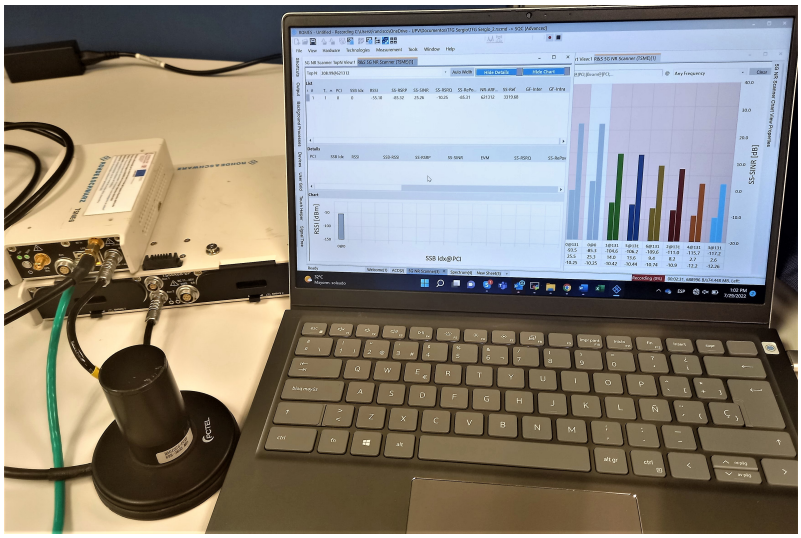


Figure 4.3: R&S *ROMES* in use.

4.1.8 Ping & Iperf

In `phy-test` mode, the `noS1` option allows to inject user traffic over the network link at the PDCP level. This makes possible the use of `ping` and `Iperf` [26] applications to test the RAN without the need for a CN. When using the `noS1` option, TUN interfaces are created at both gNB and nrUE named `oaitun_enb1` and `oaitun_ue1` with IPs `10.0.1.1` and `10.0.1.2`, respectively. They show up running the command `ip a`, similar to `ifconfig` but newer and more powerful.

Using the ping command we can send ICMP echo request and receive over this interfaces. By setting the flag -I we select the outbound interface and then we indicate the IP address of the device we are pinging. It is important to note that using this method we can assure that the packets are taking the correct route and are sent and receive with OAI because we force the use of the TUN interfaces which are internally tunnelled to the OAI nodes and we ping an IP address that is only used by these interfaces. Some MCG colleagues have had troubles in the past with sending packets between machines through the main (Internet network) gateway interface instead than through the OAI connection.

```
gNB machine: $ ping -I oaitun_enb1 10.0.1.2
UE machine:  $ ping -I oaitun_ue1 10.0.1.1
```

In addition to the ping command, we can measure the link performance using the Iperf tool. Iperf is designed in a client-server scheme with flags `-s` and `-c`. By default it's run over TCP but it can be changed to UDP with flag `-u`. The `-i` flag sets the interval time in seconds between periodic bandwidth, jitter and loss reports. The `-B` flag binds the network interface for outbound traffic at the client and for incoming traffic at the server. The `-b` flag sets the target bandwidth in bits/s. We can change the print bandwidth format with `-f` flag.

```
Server UE machine: $ Iperf -s -i 1 -u -B 10.0.1.2      - Downlink test
    -
Client gNB machine: $ Iperf -c 10.0.1.2 -u -b 100M -B 10.0.1.1

Server gNB machine: $ Iperf -s -i 1 -u -B 10.0.1.1     - Uplink test -
Client UE machine: $ Iperf -c 10.0.1.1 -u -b 100M -B 10.0.1.2
```

We must note that when using the RF Simulator the measured throughput is not the real-time one, as RFSIM can speed up or delay the transmission. Users must be warned against comparing this value with a theoretical one or to state it without a clarification as this might be misleading.

4.1.8.1 Throughput benchmark

A key parameter to measure a network is the achieved throughput. Until recently, the most important transmission direction has been Downlink as the users usually demanded much more data than uploaded. This is still relevant although new trends suggest that the Uplink is going to become more important with the upcoming technologies and uses. In this study, only the DL has been benchmarked as the TDD configuration used prioritizes this direction with more slots. A similar study can be performed for the UL.

For a performance benchmark to be relevant it has to be referred to some scale for comparison. Here, a maximum theoretical throughput has been calculated for reference. In a 5G network the throughput is determined by many parameters including the Modulation and Coding Scheme (MCS), number of sub-carriers and resource blocks used, number of symbols occupied by the DMRS, and frame duration. The standard offers the equation in Fig. 4.4 for the maximum supported data rate for DL or UL, but it does not take into consideration the number of slots scheduled in a radio frame. In the `phy-test` mode this is a relevant parameter that we can modify. Therefore, in this work I will use another frequent throughput estimator based on Transport Block Size (TBS) [27]. The TBS is the size of the payload at the PHY layer, so it includes the users' data and all the previous layers' headers but it does not include the coding and error correcting scheme. These theoretical throughputs are only estimations since in real operation there might be more overhead due to the headers and other fields added to the original data packet.

$$\text{data rate (in Mbps)} = 10^{-6} \cdot \sum_{j=1}^{J} \left(v_{Layers}^{(j)} \cdot Q_m^{(j)} \cdot f^{(j)} \cdot R_{\max} \cdot \frac{N_{PRB}^{BW(j),\mu} \cdot 12}{T_s^{\mu}} \cdot \left(1 - OH^{(j)}\right) \right)$$

Figure 4.4: 5G NR Throughput data rate equation in the DL and the UL. [28]

The TBS follows a complex determination algorithm described in [29] which is comprised of several `if` conditional statements to encompass all possible sizes. This process is shown in Fig. 4.5. For the most common sizes, including the ones used here, this conditional statements always give the same result. For this reason, I do not take into consideration all possibilities but only the ones used.

$$N_{info} = N_{RE} \cdot R \cdot Q_m \cdot \upsilon, \; N'_{RE} = N_{sc}^{RB} \cdot N_{symb}^{sh} - N_{DMRS}^{PRB} - N_{oh}^{PRB}, \; N_{RE} = \min\left(156, N'_{RE}\right) \cdot n_{PRB}$$

$$\text{If } N_{info} \le 3824, N'_{info} = \max\left(24, 2^n \cdot \left\lfloor \frac{N_{info}}{2^n} \right\rfloor\right), \text{ where } n = \max\left(3, \lfloor \log_2(N_{info}) \rfloor - 6\right)$$

$$\text{If } N_{info} > 3824, N'_{info} = \max\left(3840, 2^n \times round\left(\frac{N_{info} - 24}{2^n}\right)\right), \text{ where } n = \lfloor \log_2(N_{info} - 24) \rfloor - 5$$

$$\text{if } R \le 1/4, TBS = 8 \cdot C \cdot \left\lceil \frac{N'_{info} + 24}{8 \cdot C} \right\rceil - 24, \text{ where } C = \left\lceil \frac{N'_{info} + 24}{3816} \right\rceil$$

$$\text{else if } N'_{info} > 8424, TBS = 8 \cdot C \cdot \left\lceil \frac{N'_{info} + 24}{8 \cdot C} \right\rceil - 24, \text{ where } C = \left\lceil \frac{N'_{info} + 24}{8424} \right\rceil \text{ else } TBS = 8 \cdot \left\lceil \frac{N'_{info} + 24}{8} \right\rceil - 24$$

Figure 4.5: Transport Block Size determination algorithm. [29] [30]

To ease the TBS computing process, a MATLAB function has been written to take the same inputs as in Fig. 4.5 and output the TBS and throughput for a specified number of slots. As constant input parameters there is the number of frames per second, sub-carriers per symbol, number of symbols scheduled per PRB (in this OAI version is 13), PRB overhead, number of DMRS symbols (which for this Type and Position is 6) and number of layers. For each benchmark case we need to update the number of PRBs, modulation order, target code rate and number of slots per frame.

```
function main
    NumSlots = 1   % Number of slots per frame
    FrameDuration = 0.010 % 10ms frames
    FramesPerSecond = 1/FrameDuration
    NumSubCarriers = 12    % Number of sub-carriers per symbol
    NumSymbols = 13  % Number of symbols per PRB
    OH = 0    % PRB overhead
    NumPRB = 50    % Number of PRBs
    Qm = 6         % Modulation Order. This is for MCS 28
    R = 948        % Target code rate. This is for MCS 28
    R = R/1024;
    v = 1          % Number of Layers
    NumDMRSperPRB = 6
    NumREprime = NumSubCarriers * NumSymbols - NumDMRSperPRB - OH
    NumRE = min(156, NumREprime) * NumPRB
```

```
    NumInfo = NumRE * R * Qm * v
    if (NumInfo > 3824)
       n = floor(log2(NumInfo - 24)) - 5
       NumInfoprime = max(3840, 2^n * round( (NumInfo - 24)/(2^n) ))
       if (R > 0.25)
          if (NumInfoprime > 8424)
              C = ceil( (NumInfoprime + 24)/8424 )
              TBS_bits = 8 * C * ceil( (NumInfoprime + 24)/(8*C) ) -
                 24
          end
       end
    end
    Throughput_Mbps = TBS_bits * NumSlots * FramesPerSecond / 1000000
end
```

The MCS indexes used have been selected for being the most efficient for a given modulation order. There are three MCS tables described in the standard but the one most commonly used and implemented in OAI is Table 1 (5.1.3.1 - 1) of the standard [29] and it encompasses three modulation orders: 2, 4 and 6. A selection of table 1 for the MCS indexes chosen is available in Table 4.1.

MCS Index	Modulation Order Qm	Target code Rate R x[1024]	Spectral efficiency
9	2 (4QAM)	679	1.3262
16	4 (16QAM)	658	2.5703
28	6 (64QAM)	948	5.5547

Table 4.1: MCS index table 1 for PDSCH (Selection). [29]

4.2 FlexRIC

FlexRIC implementation of the O-RAN standard is quite new a therefore an analysis has been performed. First, the Near-RT RIC and E2 Agent at the gNB have been executed and checked the information provided by their logs. The E2 Interface between them can be inspected with Wireshark and analyzed in-depth to compare it to the standard. Wireshark's newer versions have the E2AP protocol implemented, which needs to have the port changed from 37464 to 36421. This port is the one configured by FlexRIC to send and receive E2AP messages.

4.2.1 xApps

FlexRIC was originally introduced by the MOSAIC5G project group at the Fall 2021 OpenAirInterface Workshop, where there was a dedicated Lab for xApps. The source code of the xApps was published only for the workshop and not in the GitLab repository [14]. The xApps provided by FlexRIC are related to the SM it implements, which have to do with MAC, RLC and PDCP stats.

- `helloworld.py` It connects to the RIC for some time and then disconnects.

- `monitoring.py` It subscribes to all three SMs and counts how many reports arrive. Periodically asks the API for some stats to print.
- `[mac/rlc/pdcp]_cb.py` Each one subscribes to a different SM and prints some reports received.
- `no_cb.py` It subscribes to all three SMs ignoring the reports. Makes use of the API to get information about connected UEs.

The xApps' SDK uses three different connections for the E42AP interface to the RIC: port 9990 for the initial E42Setup and pings, port 9991 for E42ReportRequests, and finally port 9992 for the E42Indication messages containing the stats. These connections are done via the WebSocket protocol. To make it work on the same machine as the RIC, the IP address has to be changed to the local interface address in file `/src/sdk/sdk_conf.py`.

```
address\_ping = "ws://127.0.0.1:9990"
address\_msg = "ws://127.0.0.1:9991"
pubsub\_address = "ws://127.0.0.1:9992"
```

Tu run the xApp's SDK over FlexRIC a Python virtual environment is created inside the SDK folder and the appropriate requirements installed. To run the xApps each time, the same commands are run except for the requirement's installation.

```
$ python3.8 -m venv ./mosaic5g
$ source ./mosaic5g/bin/activate
$ pip3 install -r requirements.txt

$ python3.8 ./xapps/helloworld.py
```

4.2.2 xApp Challenge

During the *Fall 2021 Workshop* [14], the MOSAIC5G team proposed several challenges regarding SMs and xApps, as part of their O-RAN implementation's showcase. I have focused on the xApps challenge as it is an interesting object of study for its novelty and great future possibilities as it aims to do near real-time decision making with the use of AI/ML algorithms.

There has not been any public work on these challenges and very little information has been published after the workshop, making it quite hard to even find the code as it is not included in the GitLab repository. As part of the FlexRIC analysis, studying and designing an xApp was pursued.

There are three xApp challenges proposed, each with a 500 euro prize [14]:

- **PDCP**: Compute the throughput and loss rate per UE and per Base Station for TX and RX , and visualize it over time (e.g. `matplotlib`).
- **RLC**: Compute the RLC throughput and ARQ retransmission statistics in last 1 second in terms of median, avg, deviation for TX, and visualize the m over time (e.g. `matplotlib`).

- **MAC**: Compute the aggregated throughput rate per UE/RNTI for TX and RX, and visualize it over time (e.g. `matplotlib`).

The OAI RLC mode is UM, so no re-transmission is done and therefore, the statistics provided by the RLC SM regarding this are zero-ed. Moreover, as this deployment is going to be performed with RFSIM, the PDCP stats for discarded packets that would be used to calculate the loss rate are also zero.

Finally, as all challenges mention throughput as an objective, I have decided to design an xApp to visualize the real-time Rx and Tx throughput of each layer.

4.2.3 O-RAN RIC connection

FlexRIC's implementation not only includes a near-RT RIC but also an E2 Agent that integrates with OAI's gNB to communicate over the E2Interface complying with the O-RAN standard. Therefore, the E2 Agent can speak to FlexRIC or to the O-RAN RIC implementation.

FlexRIC uses port 36421 as the E2AP server at the RIC, versus port 36422 that O-RAN standard uses. This means that the gNB E2 Agent tries to connect to a RIC at that port. To change the default port address, we can modify the source code file `FlexRIC/src/agent/e2ap_agent_api.c` (line 71) where the constant `e2ap_server_port` is defined. Then we rebuild FlexRIC. Wireshark's E2AP protocol port has to be changed also to view these messages.

The RIC's IP address can also be modified in case the gNB and near-RT RIC are not in the same machine. To do so, simply change the value in `FlexRIC/FlexRIC.conf`, without the need to rebuild.

The O-RAN RIC had been already deployed at the iTEAM Lab on Kubernetes. On this configuration, the RIC cluster does not publicly expose the e2term SCTP endpoint, and so a workaround is to create a socat to route from one location to another. This can be done with the following script:

```
#!/bin/bash

if [ $# -ne 1 ]; then
echo "need one parameter"
exit
fi

killall socat
set -x
ip=10.97.194.53          - Kubernetes e2term's IP address -

for (( c=0; c<$1; c++ )); do
let port=36421+$c
nohup socat SCTP4-LISTEN:$port SCTP4:$ip:36422 &
done

netstat -tulpnS
```

When a E2 Agent is connected to the e2term node at the RIC, the e2mgr node will show the E2Setup procedure and some information about the connected gNB's can be retrieved via an API request.

Chapter 5

Results and Discussion

In this chapter, the results obtained using the setup and tools described in chapters 3 and 4 are exposed and discussed. It begins with the OAI RAN deployment and execution, explaining different modes and why a RF simulator had to be used instead of the USRPs, and conducting a throughput benchmark. Next comes the FlexRIC analysis and an explanation of the xApp designed. Finally, a newer OAI RAN version is tested with USRPs, also measuring is throughput.

5.1 OAI RAN

The aim of OAI RAN is to create a wireless communication link between to machines to allow data to be transmitted. During the deployment of the OAI RAN, some problems where encountered and so different modes and radio heads are used, all of which is described below.

5.1.1 Stand-Alone mode

When executing in SA mode, the UE follows the random access procedure by decoding the MIB and SIB1 and sending the RRC Setup Request to arrive to UE State `NR_RRC_CONNECTED`. But as no Core has been linked in this RAN deployment, the gNB cannot connect with the AMF and so it cannot proceed with the UE attachment. Thus, this mode of operation is not suitable for this RAN study, as expected. Nonetheless, some valuable information can still be extracted from the SA analysis from the gNB's Broadcast Channel (BCH) and the incomplete UE attachment procedure.

It must be noted that this test in SA mode is performed using USRPs and they are able to successfully communicate over-the-air (OTA), this will be important later.

The gNB and UE logs show the message exchange for the attachment procedure, indicating that it is not possible to successfully finish it because no AMF is present. The process goes as follows: First, the UE detects sync, decodes the MIB, SIB1 and sends RA message 1. Next, the gNB sends message 2 with the preamble index the UE is expecting. The UE finds this value and transmits message 3. Then the gNB sends message 4 with the RNTI to conclude the RA procedure and changes to state `NR_RRC_CONNECTED`. Finally, the UE sends a RRCSetupComplete message to arrive at the Core, but the gNB informs that it cannot arrive as no AMF is associated.

```
- UE log -

[PHY]   [SCHED][UE] Check absolute frequency DL 3319680000.000000, UL 3319680000.000000 (RF card 0, oai_exit 0,
     channel 0, rx_num_channels 1)
[PHY]   Starting sync detection
[PHY]   [UE thread Synch] Running Initial Synch (mode 6)
[PHY]   [UE] nr_synchro_time: Sync source = 0, Peak found at pos 4668, val = 1225231225 (90 dB) avg 63 dB, ffo
     0.000000
PSS execution duration 336673 microseconds
[PHY]   [UE0] Initial sync : Estimated PSS position 4668, Nid2 0
[PHY]   sync_pos 4668 ssb_offset 4524
[PHY]   Calling sss detection (normal CP)
[PHY]   [UE0] Initial sync: starting PBCH detection (rx_offset 0)
[NR_RRC]   Configuring MAC for first MIB reception
[PHY]   [UE0] Initial sync: pbch decoded sucessfully
[PHY]   TDD Normal prefix: CellId 0 metric 18590, phase 3, pbch 0
[PHY]   [UE0] In synch, rx_offset 108 samples
[PHY]   [UE 0] RRC Measurements => rssi -inf dBm (dig -inf dB, gain 40), N0 0 dBm,  rsrp -inf dBm/RE, rsrq -inf dB
[PHY]   [UE 0] Measured Carrier Frequency 3319680000 Hz (offset 0 Hz)

[NR_RRC]   SIB1 decoded
[NR_MAC]   NR band duplex spacing is 0 KHz (nr_bandtable[37].band = 78)
[NR_MAC]   NR band 78, duplex mode TDD, duplex spacing = 0 KHz
[MAC]   Setting TDD configuration period to 6
[PHY]   TDD has been properly configurated
[MAC]   Initializing ul_config_request. num_slots_ul = 3
[NR_RRC]   [UE 0] : Logical Channel UL-CCCH (SRB0), Generating RRCSetupRequest (bytes 6, gNB 0)
[PHY]   Resynchronizing RX by 108 samples (mode = 6)
[MAC]   Initialization of 4-step contention-based random access procedure
[PHY]   PRACH [UE 0] in slot 19, placing PRACH in position 2828, msg1 frequency start 0 (k1 0), preamble_offset 1,
     first_nonzero_root_idx 0

[NR_MAC]   [UE 0][RAPROC] Got BI RAR subPDU 5 ms
[NR_MAC]   [UE 0][RAPROC] Got RAPID RAR subPDU
[NR_MAC]   [UE 0][RAPROC][914.7] Found RAR with the intended RAPID 8
[NR_MAC]   [RAPROC][914.17] RA-Msg3 transmitted

[MAC]   [UE 0][RAPROC] Frame 916 : received contention resolution identity: 0x1cd7c1e54726 Terminating RA procedure
[MAC]   [UE 0][916.1][RAPROC] RA procedure succeeded. CB-RA: Contention Resolution is successful.
<DL-CCCH-Message></DL-CCCH-Message>
[NR_RRC]   [UE0][RAPROC] Frame 916 : Logical Channel DL-CCCH (SRB0), Received NR_RRCSetup RNTI e40a
[RRC]   Received mac_CellGroupConfig from gNB
[MAC]   Applying CellGroupConfig from gNodeB
[NR_RRC]   [UE 0], CONFIG_SRB1 1 corresponding to gNB_index 0
[NR_RRC]   [FRAME 00916][RRC_UE][MOD 00][][--- MAC_CONFIG_REQ  (SRB1 gNB 0) --->][MAC_UE][MOD 00][]
[NR_RRC]   [UE 0] State = NR_RRC_CONNECTED (gNB 0)
[CONFIG] uicc0: 9/9 parameters successfully set
[SIM]   UICC simulation: IMSI=2089900007487, Ki=fec86ba6eb707ed08905757b1bb44b8f, OPc=c42449363bbad02b66d16bc975d77cc1
[NR_RRC]   [UE 0][RAPROC] Frame 916 : Logical Channel UL-DCCH (SRB1), Generating RRCSetupComplete (bytes41, gNB 0)
[NR_MAC]   [924.1] Received TA_COMMAND 32 TAGID 1 CC_id 0

- gNB log -

[NR_PHY]   [gNB 0][RAPROC] Frame 913, slot 19 Initiating RA procedure with preamble 8, energy 54.0 dB (I0 82, thres
     120), delay 13 start symbol 0 freq index 0
[NR_MAC]   [gNB 0][RAPROC] CC_id 0 Frame 914, slotP 7: Generating RA-Msg2 DCI, rnti 0x010b, state 1, CoreSetType 2
[NR_MAC]   [gNB] Generate RAR MAC PDU frame 914 slot 7 preamble index 8 TA command 13
[NR_MAC]   [gNB 0] Adding UE with rnti 0xe40a (num_UEs 0)
[NR_MAC]   [gNB 0][RAPROC] PUSCH with TC_RNTI 0xe40a received correctly, adding UE MAC Context UE_id 0/RNTI 0xe40a
[NR_MAC]   [RAPROC] RA-Msg3 received (sdu_lenP 8)
[NR_RRC]   [FRAME 00914][gNB][MOD 00][RNTI e40a] [RAPROC] Logical Channel DL-CCCH, Generating RRCSetup (bytes 106)
[NR_MAC]   Scheduling RA-Msg4 for TC_RNTI 0xe40a (state 4, frame 916, slot 1)
[NR_MAC]   [gNB 0] [RAPROC] CC_id 0 Frame 916, slotP 1: Generating RA-Msg4 DCI, state 4
[NR_MAC]   (ue 0, rnti 0xe40a) Received Ack of RA-Msg4. CBRA procedure succeeded!
[NR_RRC]   Received message NR_RRC_DCCH_DATA_IND
[NR_RRC]   Received rrcSetupComplete, 5g_s_TMSI: 0x123456789ABC, amf_set_id: 0x48(72), amf_pointer: 0x34(52), 5g TMSI:
     0x56789ABC
[NR_RRC]   [FRAME 00000][gNB][MOD 00][RNTI e40a] [RAPROC] Logical Channel UL-DCCH, processing NR_RRCSetupComplete from
     UE (SRB1 Active)
[NR_RRC]   [FRAME 00000][gNB][MOD 00][RNTI e40a] UE State = NR_RRC_CONNECTED
[NGAP]   No AMF is associated to the gNB
```

5.1.1.1 ROMES

For the UE to initiate the attachment procedure it has to first sync and decode the MIB and SIB1 that the gNB is broadcasting. To check the gNB's BCH, R&S *ROMES 4* is available. In it we can see all 5G Base Stations (BS) broadcasting in a desired spectrum. Where iTEAM is located, there are several mobile network operators with 5G BS that show up on *ROMES*. In Fig. 5.1, we can locate our gNB on the left with MCC 208 (France) and MNC 99, and Orange and Vodafone to the right. They both have Spanish MCC 214. In the middle there is another carrier with MNC 09 that belongs to Orange although *ROMES* does not recognise it.

Figure 5.1: R&S *ROMES 4* showing the 5G NR spectrum.

Operators assign a different Physical Cell Identifier (PCI) to differentiate their BS when more than one are received simultaneously. Although our BS is composed of only one beam, it is common for operators to set up several at a single BS and differentiate them by the SSB Index. *ROMES*'s scanner displays several physical parameters for each beam in a BS in the form SSB Idx@PCI. This information is shown in Fig. 5.2, where our beam 0@0 has the highest Reference Signal Received Power (RSRP) of all, at -78.9dBm. It is not shown as the preferred beam (most to the left in the right diagram) because it has a lower Signal-to-Interference-plus-Noise Ratio (SINR) than 0@131 at 26dBm.

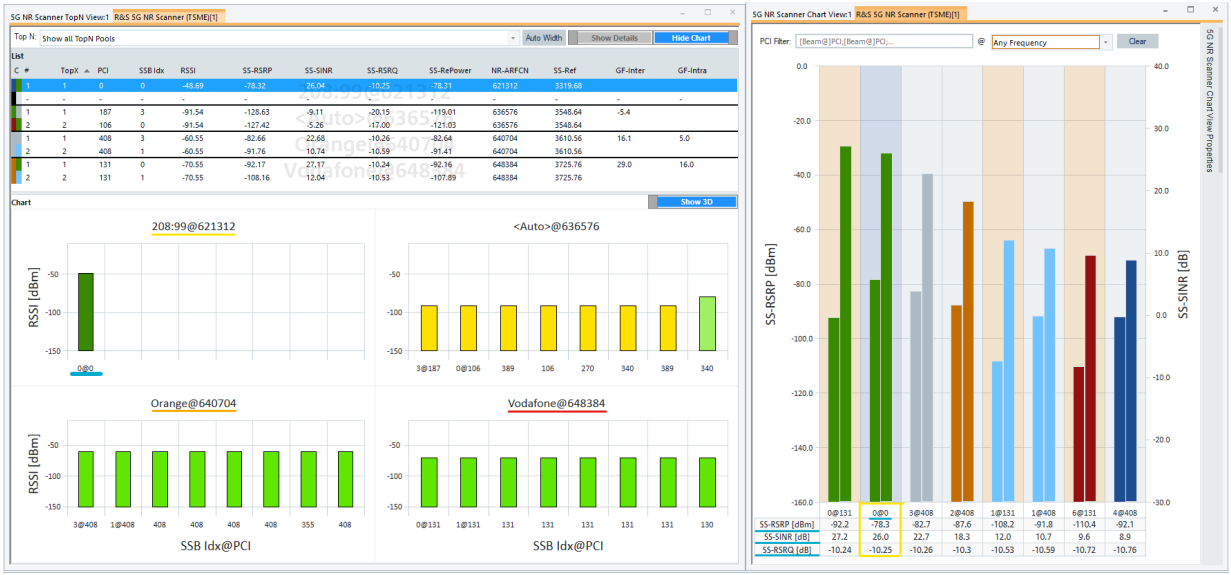
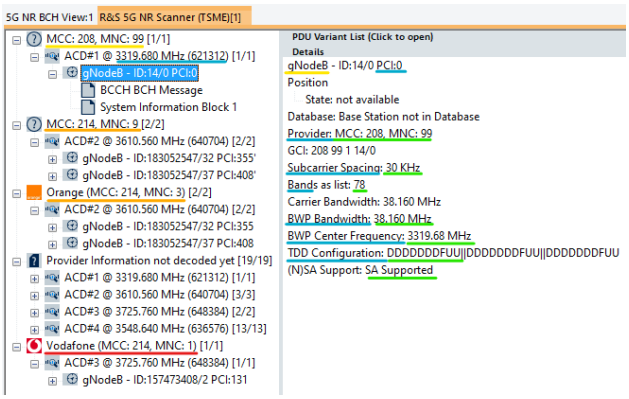


Figure 5.2: R&S *ROMES 4* showing the 5G NR scanner.

The big difference between *ROMES* and a spectrum analyzer is that the former can decode the BCH's information. Fig. 5.3 shows the parameters broadcasted by the gNB. First, the gNB info is a summary provided by the program with information from the MIB and SIB1. It shows the PCI, the providers' MCC and MNC, the SCS, band, bandwidth, carrier frequency, TDD configuration and SA support. The TDD configuration highlighted is very use full as it shows Downlink, Flexible and Uplink slots in a frame. Our frames are 5ms long and contain 10 slots, and this format is a very comprehensive way to understand it.

The MIB info is serves to help the UE locate the SIB1 and are not main parameters of the gNB. Please note that the intra-Frequency Reselection (i.e. handover) is not allowed while the operators' BS do allow it.

The full SIB1 information is available in Fig. 5.4 where MCC and MNC are specified, along with the band, SCS and bandwidth for DL and UL. It also indicates the TDD configuration with its periodicity, DL and UL slot allocation, and DL and UL symbols in the mixed slot.



(a) gNB info **(b)** MIB info

Figure 5.3: R&S *ROMES 4* showing the SA BCH info.

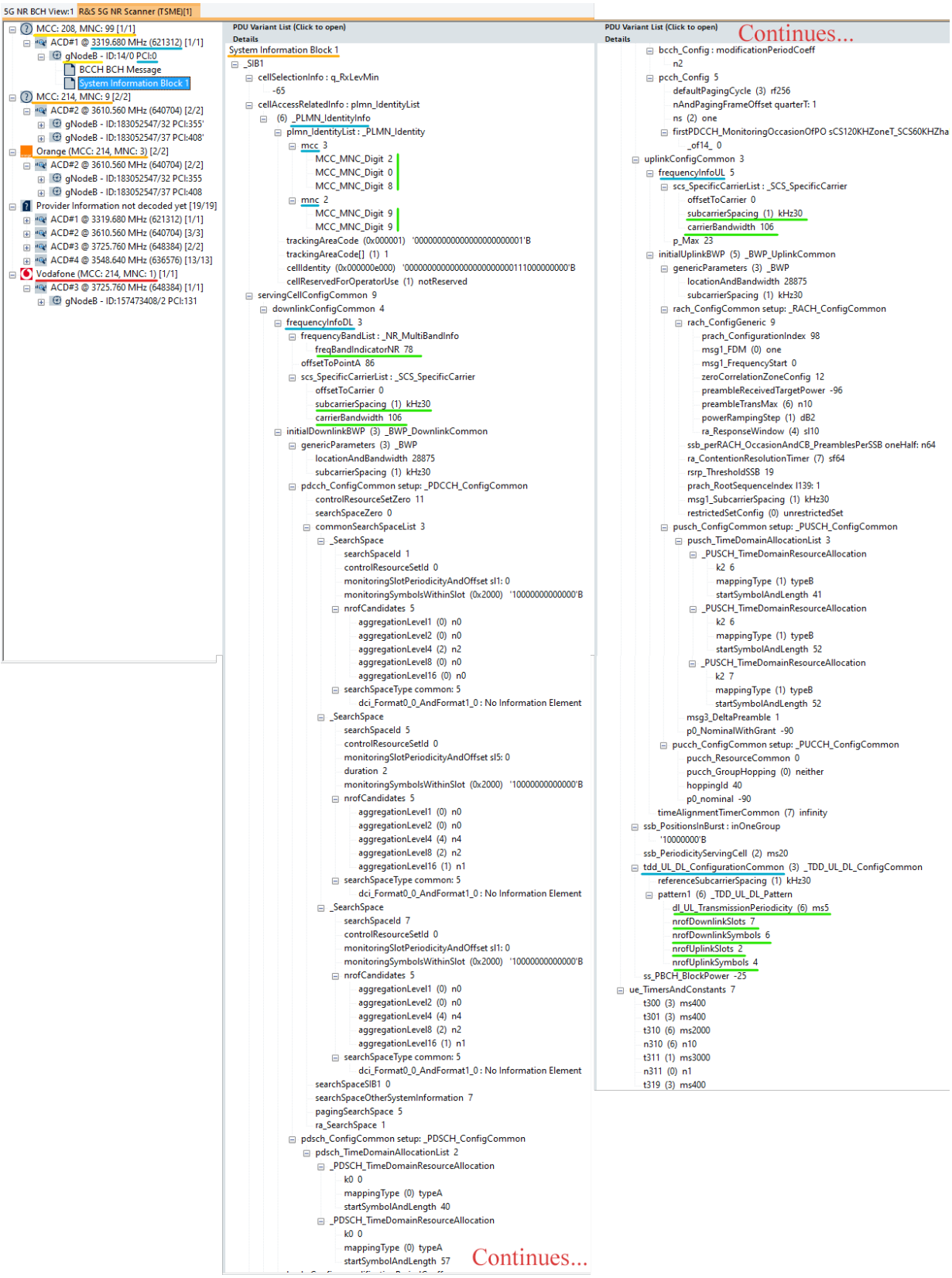


Figure 5.4: R&S *ROMES 4* showing the SA SIB1 info.

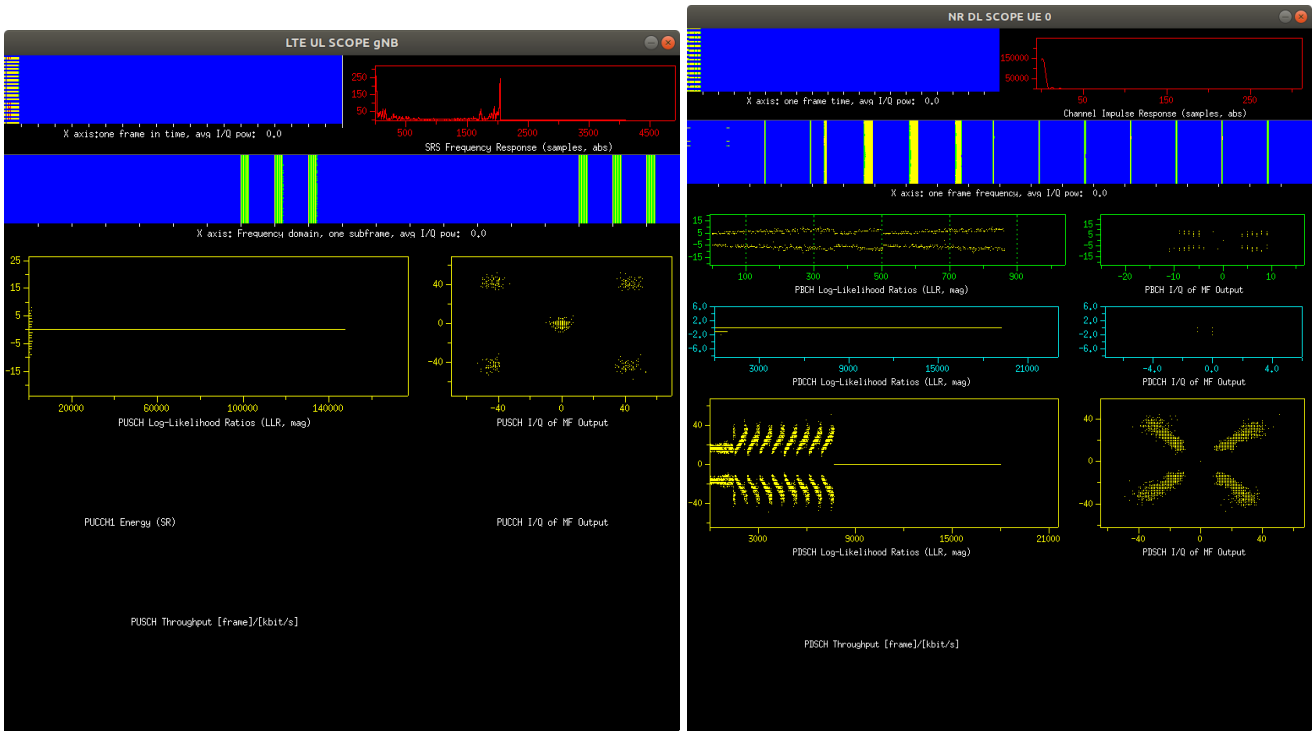
5.1.2 Phy-Test mode

Since the scope of this project does not include deploying a Core Network, the OAI option to go to would be the `phy-test`. In this mode the UE attachment is simulated and the MIB, SIB1 and RRC communication is replaced with two files from which this information is read.

The gNB log shows the UE RNTI 1234 that the `phy-test` mode auto-schedules and some parameters as the number of ulsch rounds and errors or the ulsch total bytes received.

```
[NR_PHY]    Number of bad PUCCH received: 460
[NR_MAC]    Frame.Slot 512.0
UE ID 0 RNTI 1234 (1/1) PH 0 dB PCMAX 0 dBm, average RSRP -88 (8 meas)
UE 0: dlsch_rounds 1189/116/115/115, dlsch_errors 115, pucch0_DTX 460, BLER 0.0000 MCS 0
UE 0: dlsch_total_bytes 1485061
UE 0: ulsch_rounds 387/385/382/381, ulsch_DTX 1529, ulsch_errors 381
UE 0: ulsch_total_bytes_scheduled 483363, ulsch_total_bytes_received 6245
UE 0: LCID 4: 675 bytes TX
```

When running in this mode with USRPs, the UL channel is able to send some bytes to be received by the gNB, but suddenly freezes and stops increasing just as the gNB scope shows stops updating the PUSCH constellation. The number of bytes received by the gNB are logged as `ulsch_total_bytes_received` and are a different number in each run and always very few. In the UE Scope, we can see as the PDSCH and PDCCH constellations freezes after a few transmissions although the PBCH constellation keeps updating. After this moment, the `ulsch_errors` increase continuously.



(a) gNB Scope **(b)** UE Scope

Figure 5.5: Execution with phy-test mode over USRPs

5.1.2.1 Spectrum Analyzer

To aid debugging this situation, the spectrum analyzer is used to check for an adequate DL physical transmission. The R&S *FSW*'s screenshot in Fig. 5.6 shows that the analyzer is able to sync with the gNB and show its PBCH 4QAM constellation diagram.

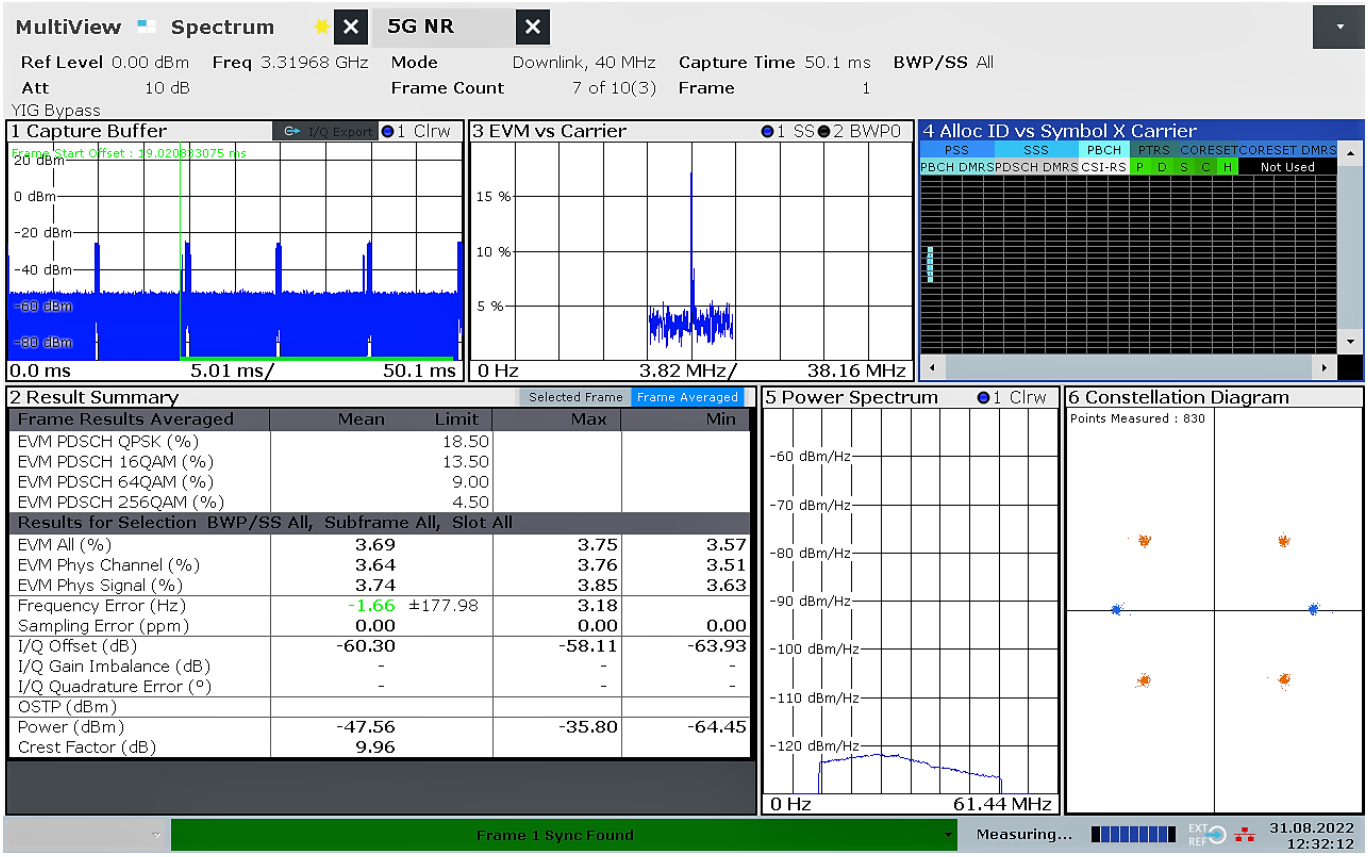


Figure 5.6: R&S *FSW* spectrum analyzer showing the gNB DL capture.

When asked to the developers they suggested increasing the pusch target SNR value in the config file in case it is a power control issue but it does not solve the problem. Adjusting the gains also does not result in substantial changes. This behaviour prevents communication using USRPs in this version, and forces the use of a radio head alternative, to which OAI offers the RF Simulator. Newer OAI versions do allow for USRP usage so we can conclude there is some kind of bug in the current version.

5.1.3 RF Simulator

Thus, the RF Simulator has to be used instead of the USRPs. In this version other users have experimented issues when using radio heads and it could be due to an OAI bug, as it is mainly tested with RFSIM during development.

Using the `phy-test`, `rfsimulator` and `noS1` option, we are able to ping between the UE and gNB.

5.1.3.1 Wireshark packet sniffing

Having finally established an adequate communication, we can analyse the messages transmitted over the link using OAI's T-tracer and Wireshark.

First, in Fig. 5.7 a general view of the default `phy-test` scheduling can be seen. In it the MIB inside the SSB is periodically transmitted every 20ms, and in between are two 10ms frames. In each frame up to 20 slots can be scheduled but in the default configuration only the first for DL and UL is used. If more slots are configured using a bitmap and option -D or -U then more UL and DL messages will lay between each MIB. In this Fig. several layer's protocols are shown: RRC for MIB, MAC and RLC. MAC messages indicate the transport channel used : ULSCH or DLSCH. RLC messages indicate the direction and AM mode, as well as the control-plane and ACK information.

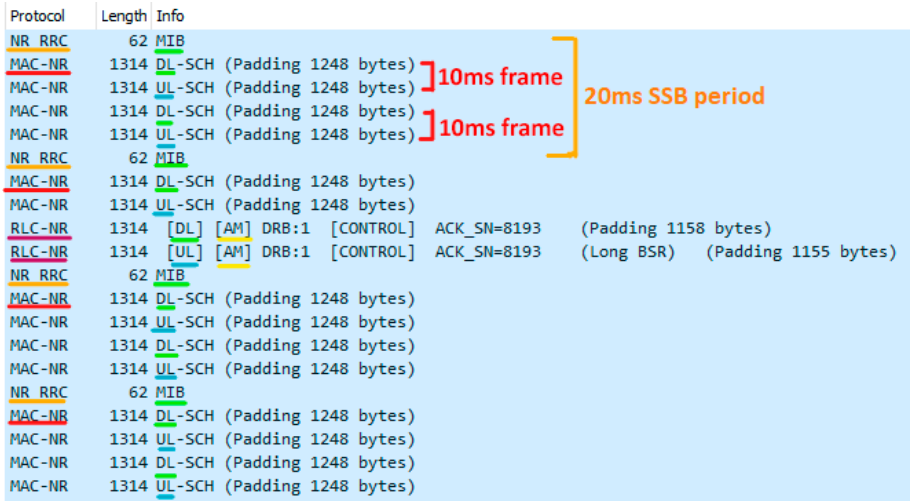


Figure 5.7: *Wireshark* OAI Tracer's capture showing a `phy-test` idle sequence.

In Fig. 5.8 we focus on the MIB message's details where the different protocol's headers are decoded. Some relevant information shown is the duplex mode TDD, direction, frame number and slot number, together with the actual MIB information.

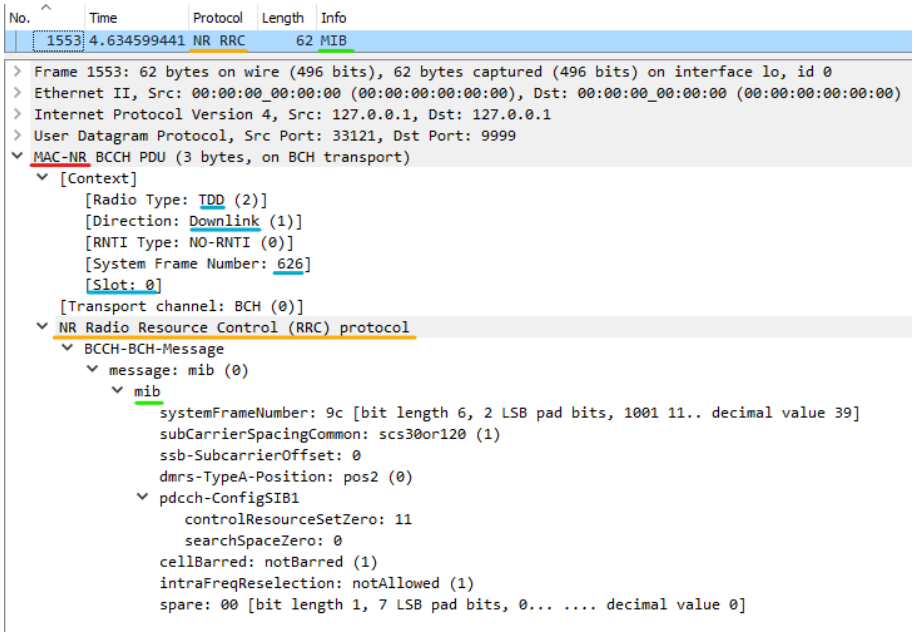
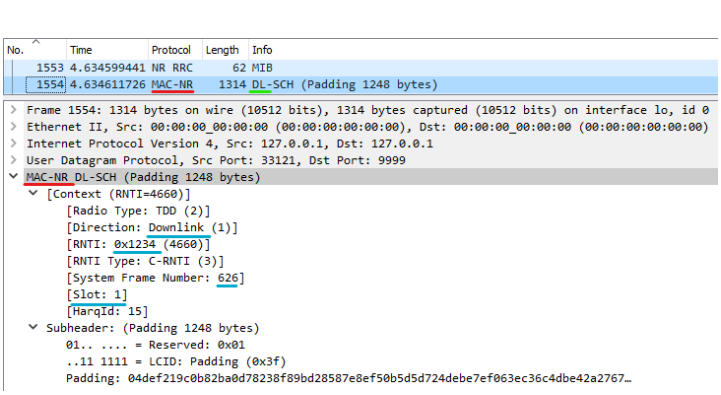
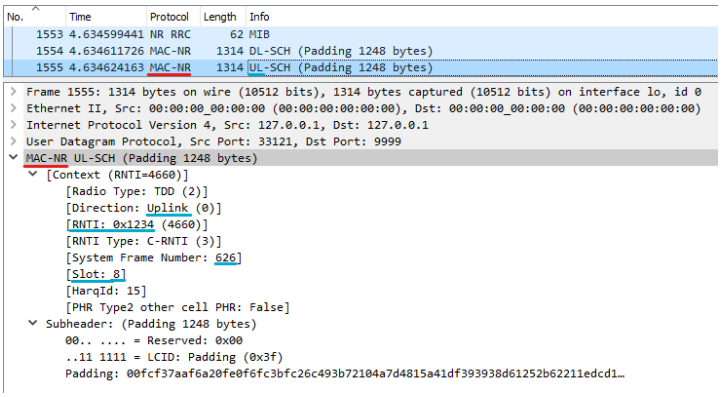


Figure 5.8: *Wireshark*'s OAI Tracer capture showing a MIB message.

Fig. 5.9 details two MAC messages in both directions. We can highlight the default RNTI used by the `phy-test` as well as the slot numbers. These two MAC messages and the previous MIB are part of frame number 626. The MIB and DL MAC messages utilize the first two DL slots (0 and 1) and the UL MAC message utilizes the first UL slot (8). Based on the TDD configuration used, the 10 slots TDD pattern repeats every 5ms and two repetitions fill up a frame. For this reason, 20 slots can be scheduled in each frame. Here only the first TDD repetition is used but later, when performing the throughput benchmarks, up to 12 slots will be used in each frame.



(a) Down-link MAC message



(b) Up-link MAC message

Figure 5.9: *Wireshark*'s OAI Tracer capture showing two MAC messages' details.

Apart from the continuously scheduled MAC messages in `phy-test` mode, a higher layer ping message can be seen in Fig. 5.10. In it all three layer's headers are shown, indicating that it is an UL message, the RLC mode is AM containing data and it is user-plane information. Finally, the data payload shows an incremental 2-bytes blocks that form the ping packet. Instead of being random data it is sequential and some 16bit blocks can be decoded as ASCII text.

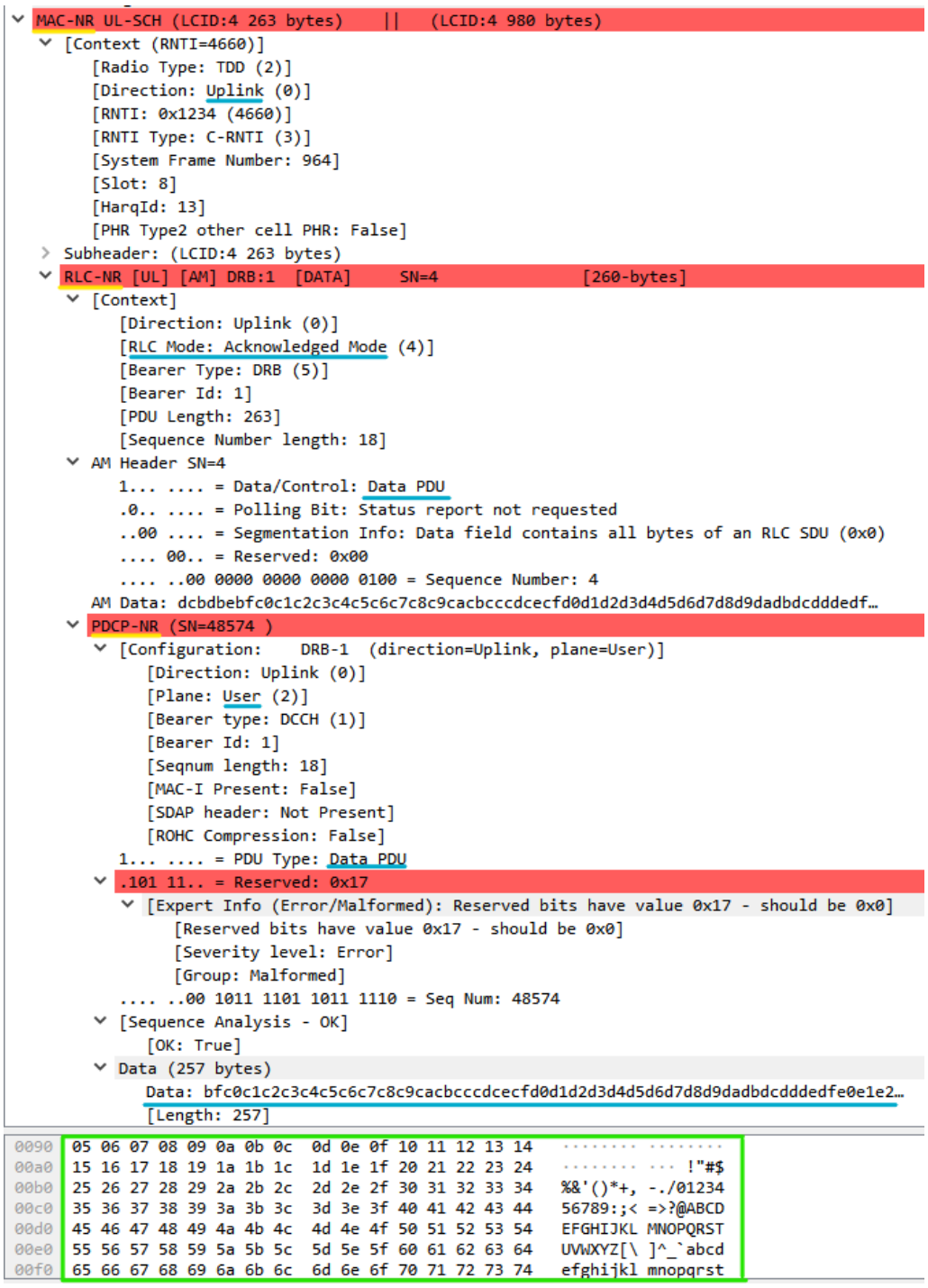


Figure 5.10: *Wireshark* OAI Tracer's capture showing a UL ping message.

5.1.3.2 Throughput benchmark

The Iperf performance tests done for this benchmark with the RF Simulator do not reflect the true real-time throughput as would be with USRPs but it is still interesting and has been performed in [27]. These benchmarks are performed with Iperf for different MCSs, PRBs and number of slots.

The Iperf benchmarks have been performed over UDP with the server located at the UE and the client at the gNB, for 60 seconds at a target bandwidth of 100Mbps. The UE server log's results are available in Table 5.1 along with the theoretical throughput in the format: [real/theoretical]. The measured jitter, or latency variation, for each test has also been recorded.

DL throughput in Mbps						
	50 PRBs			106 PRBs		
# of slots	1	6	12	1	6	12
MCS 9	0.46 / 1.00	2.78 / 6.00	4.98 / 11.99	0.98 / 2.10	5.81 /12.60	9.15 / 25.20
MCS 16	0.91 /1.95	5.39 / 11.68	8.51 / 23.36	1.91 / 4.10	10.3 / 24.59	14.4 / 49.17
MCS 28	1.93 / 4.20	11.7 / 25.21	18.0 / 50.42	4.11 / 8.81	21.0 / 52.84	28.4 / 105.68
DL jitter in ms						
	50 PRBs			106 PRBs		
# of slots	1	6	12	1	6	12
MCS 9	25.395	4.381	2.247	11.916	1.835	1.154
MCS 16	16.120	2.477	1.313	6.666	1.453	0.948
MCS 28	6.338	0.783	0.974	3.533	1.236	0.564

Table 5.1: RF Simulator UDP throughput benchmark

Together with the Iperf throughput benchmark, ping tests have been performed to measure latency for a 56 bytes packet (84 bytes with headers). The latency is similar for every configuration at around 25 mili-seconds round trip. This constant nature is due to the small size of the packet that fits in a single slot in every run and thus it does not benefit from the increased TBS. If we would have done this test with a higher packet size, it would have splitted into several slots in some configurations and would not offer a more accurate latency estimation as it would be more like a throughput estimation.

In Fig. 5.11 a comparison between the measured throughput and the theoretical throughput is displayed. The diagonal line represents a measured throughput equal to the theoretical one and so, if the points are closer to it, it means there is less difference between them. In this case, as the RF Simulator is not processing in real-time, the measured throughput is significantly lower than the theoretical one and diverges more as more computing power is needed for more complex configurations.

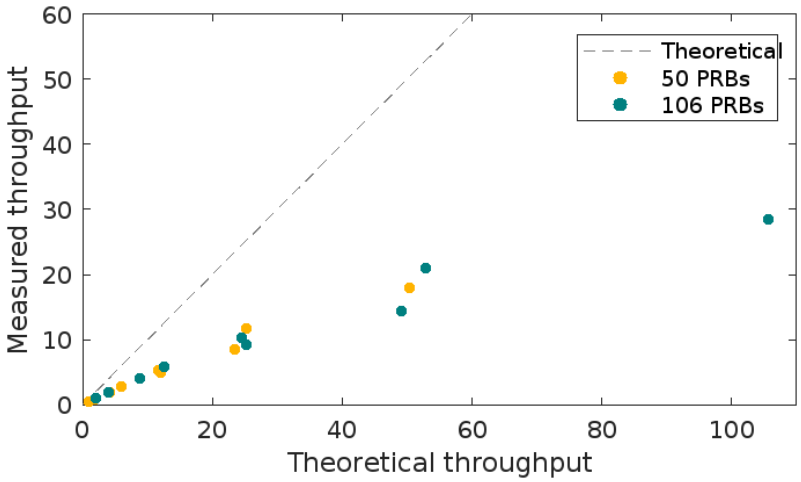


Figure 5.11: RF Simulator theoretical throughput vs. measured throughput

5.2 FlexRIC

5.2.1 Near-RT RIC

The `near_ric_sa` executable is FlexRIC's implementation of the Near-RT RIC, and when executed it waits for an incoming E2AP setup request. When the gNB is started, it automatically sets up the E2AP interface with the RIC. The RIC in turn subscribes to the SM RAN Functions to receive periodic reports.

```
                - gNB -
[E2 NODE]: mcc = 208 mnc = 99 mnc_digit = 2 nd_id = 3584
[E2 AGENT]: RIC IP Address = 127.0.0.1
[E2 AGENT]: Initializing ...
[E2 AGENT]: Opening plugin from path = /usr/lib/FlexRIC/libmac_sm.so
[E2 AGENT]: Opening plugin from path = /usr/lib/FlexRIC/libslice_sm.so
[E2 AGENT]: Opening plugin from path = /usr/lib/FlexRIC/libpdcp_sm.so
[E2 AGENT]: Opening plugin from path = /usr/lib/FlexRIC/librlc_sm.so
[E2AP] Sending setup request
                - RIC -
[NEAR RIC]: RIC IP Address = 127.0.0.1
[NEAR-RIC]: Initializing
[NEAR-RIC]: Loading SM ID = 142 with def = MAC_STATS_V0
[NEAR-RIC]: Loading SM ID = 145 with def = SLICE_STATS_V0
[NEAR-RIC]: Loading SM ID = 144 with def = PDCP_STATS_V0
[NEAR-RIC]: Loading SM ID = 143 with def = RLC_STATS_V0
[E2AP] Received SETUP-REQUEST from PLMN 208.92 Node ID 42
[NEAR-RIC]: Accepting RAN function ID 142 with def = MAC_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 143 with def = RLC_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 144 with def = PDCP_STATS_V0
[NEAR-RIC]: Accepting RAN function ID 145 with def = SLICE_STATS_V0
                - gNB -
[E2-AGENT]: SETUP-RESPONSE received
                - RIC -
[E2AP] SUBSCRIPTION REQUEST generated
[E2AP] SUBSCRIPTION RESPONSE received
[iApp]: Server ping started
[iApp]: Server Request/Reply started
```

The RIC saves a log file `log.txt` with the received stats from the RIC Indication reports. Even if the data could be retrieved from this file to be analyze, the preferred way is through the xApps. The reported data is stored as shown below:

```
mac_stats: tstamp=1660813571933961,dl_aggr_tbs=2346,ul_aggr_tbs=1853, ...
rlc_stats: tstamp=1660813571934203,txpdu_pkts=2346,txpdu_bytes=1853, ...
pdcp_stats: tstamp=1660813571934430,txpdu_pkts=83,txpdu_bytes=194, ...
slice_stats: tstamp=1660813571934635
```

It has to be noted that this log is only saved on the first attachment of the gNB to the RIC, so the RIC must be stopped and re-executed each time the gNB runs. This also happens with the xApps as, even though the RIC accepts multiple gNB setups, only the first one can communicate with the xApps.

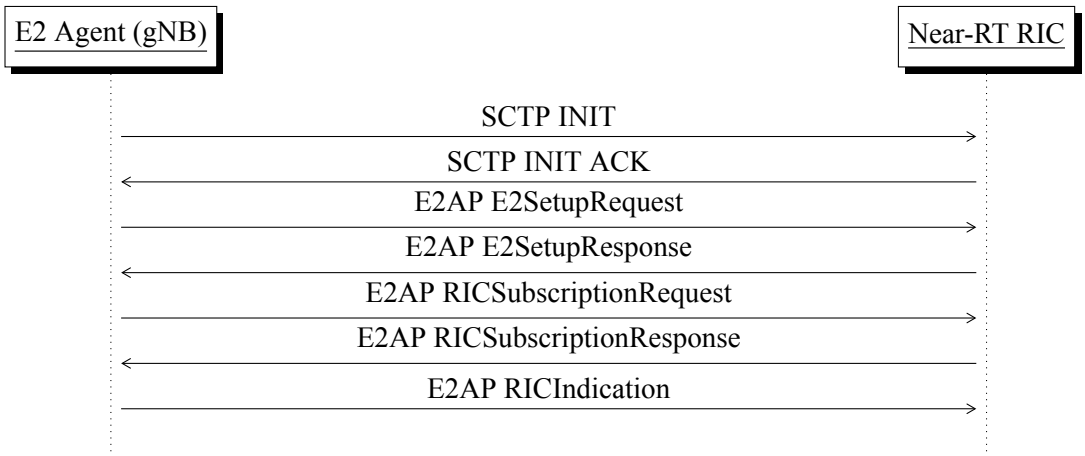
5.2.1.1 Wireshark packet sniffing

The E2 Interface messages can be captured using Wireshark on the loopback interface of the machine (127.0.0.1) , as both the gNB and RIC are running on it. The E2 Agent in the gNB open a random SCTP port to stablish communication with the RIC at the fixed port 36421. The E2AP setup, the following RIC subscriptions sequence, and the RIC indication messages captured by Wireshark can be seen in Fig. 5.12.

Source port	Destination port	Protocol	Length	Info
52926	36421	SCTP	146	INIT
36421	52926	SCTP	370	INIT_ACK
52926	36421	E2AP	478	COOKIE_ECHO , E2setupRequest[Malformed Packet]
36421	52926	SCTP	66	COOKIE_ACK SACK (Ack=0, Arwnd=62499883)
36421	52926	E2AP	122	E2setupResponse
52926	36421	SCTP	62	SACK (Ack=0, Arwnd=62499940)
36421	52926	E2AP	102	RICsubscriptionRequest
52926	36421	E2AP	118	SACK (Ack=1, Arwnd=62500000) , RICsubscriptionResponse
36421	52926	E2AP	174	SACK (Ack=1, Arwnd=62499962) , RICsubscriptionRequest, RICsubscriptionRequest
52926	36421	E2AP	118	SACK (Ack=3, Arwnd=62499960) , RICsubscriptionResponse
52926	36421	E2AP	122	RICindication
36421	52926	SCTP	62	SACK (Ack=3, Arwnd=62499942)
52926	36421	E2AP	102	RICsubscriptionResponse
52926	36421	E2AP	122	RICindication
36421	52926	SCTP	62	SACK (Ack=5, Arwnd=62499904)
52926	36421	E2AP	122	RICindication
52926	36421	E2AP	122	RICindication
36421	52926	SCTP	62	SACK (Ack=7, Arwnd=62499942)

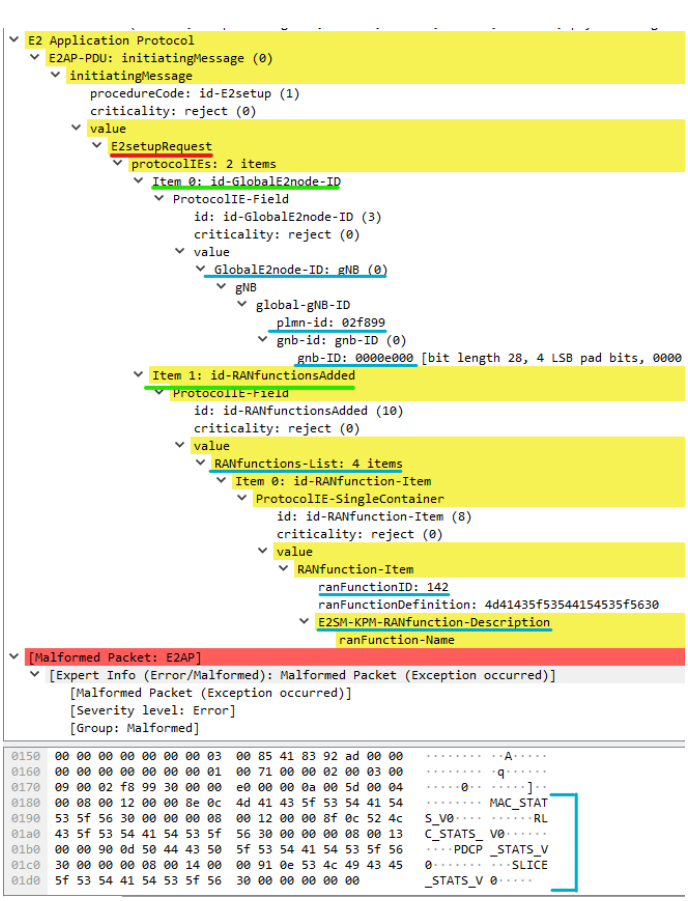
Figure 5.12: E2AP Setup and RIC subscription sequence

The exchange of packets, as defined by the standard, is reproduced in the following sequence diagram:



E2SetupRequest The packet informs the RIC about which type of E2 Node it is (gNB), the plmn (2f899) and ID (e000), as well as indicating a list of RAN Functions IDs (142), names and E2SM. This packet is shown in Wireshark as Malformed although it is properly decoded by the E2 Node, and the raw data shows the 4 RAN Function names.

E2SetupResponse The setup response informs about the plmn, RIC ID and the list of RAN Functions accepted, which in this case are all four requested: `MAC_STATS_V0`, `RLC_STATS_V0`, `PDCP_STATS_V0` and `SLICE_STATS_V0`. It must be noted that the Slice SM is not implemented in this version.



(a) E2SetupRequest

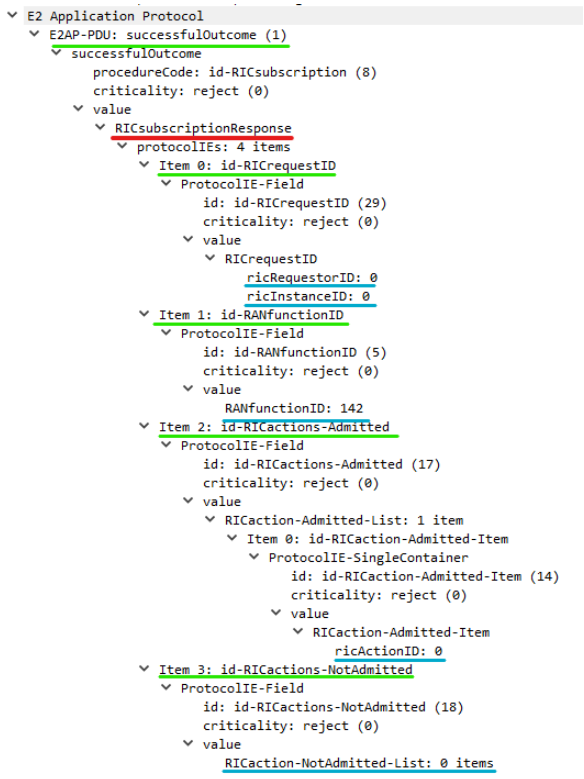
(b) E2SetupResponse

Figure 5.13: E2AP Setup packet details

RICSubscriptionRequest The subscription packet must inform the ID of the RIC who sends it, a Requestor ID,the RAN Function it wants to subscribe to, the Event Trigger definition and the type of action that the E2 Node should perform when triggered. In this case, RIC `0` is subscribing to RAN Function `142` (`MAC_STATS_V0`) to receive a periodic (event trigger: 0) report (action type: 0). The subscription request is done for each RAN function the RIC wants to get reports from.

RICSubscriptionResponse The subscription response communicates the admitted RIC actions for the RAN function requested. After this response, the E2 Node will start with its procedure of waiting for an event trigger and sending a `RICIndication` message with the pre-defined report to the RIC.

(a) RICSubscription Request



(b) RICSubscription Response

Figure 5.14: RIC Subscription packet details

RICIndication The indication messages report to the RIC the information pre-defined by the SM. It indicates the RIC ID and unique Requestor ID for each subscription, the RAN Function ID, the indication type, the indication header and the data message.

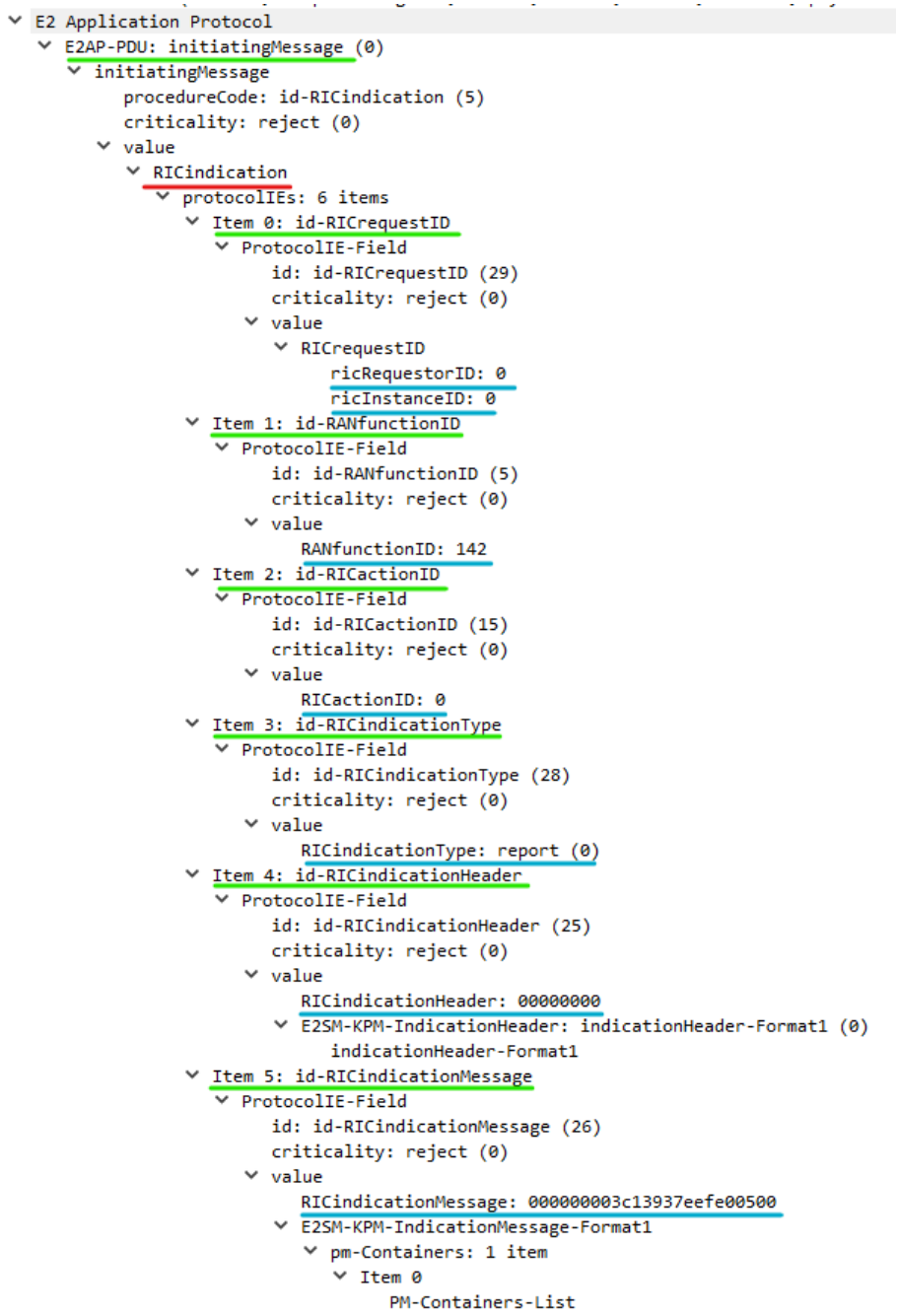


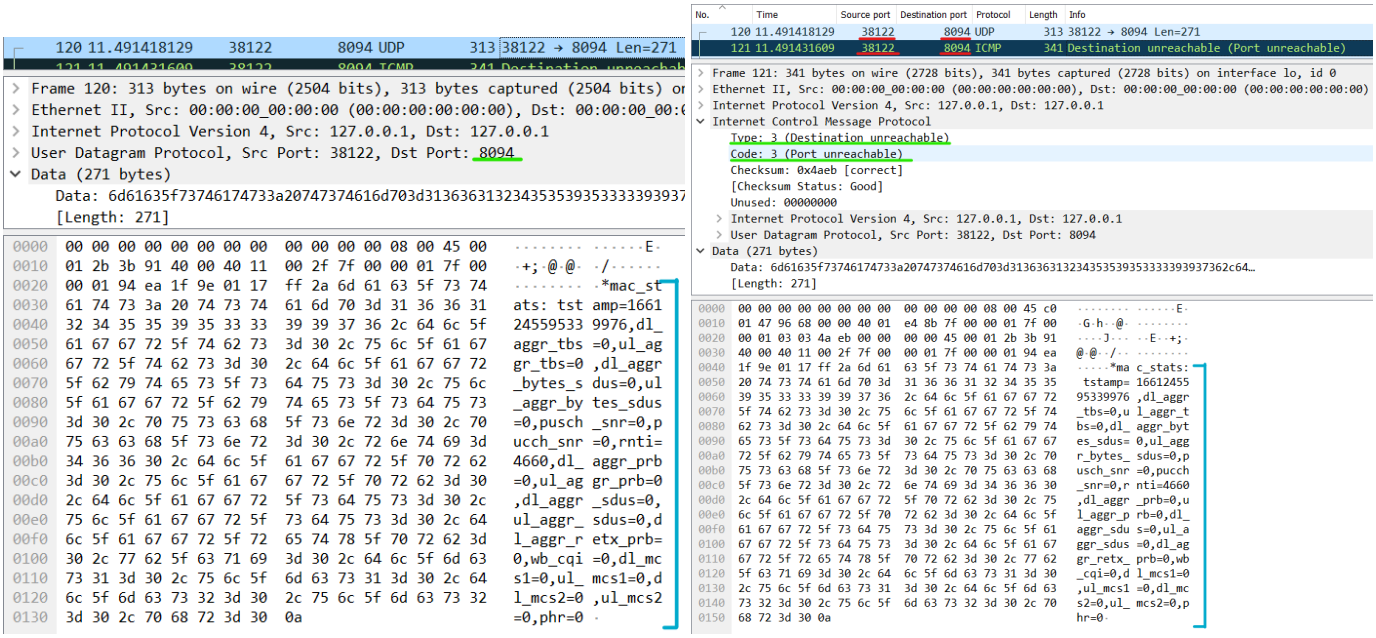
Figure 5.15: RIC Indication packet details

SM iApp Indication Focusing on the FlexRIC SDK structure shown in Fig. 2.12, we can appreciate the distinction between the *Server library* and the *iApps*. On the former one lays the E2 Server where the RICIndication messages are received. On the latter, are located the SM iApps that handle the information provided by these indications. The connection from the E2 Server to the SM iApps goes through the loopback interface towards the iApp influx UDP port 8094. For each RICIndication message there is an iApp message carrying the decoded text information in raw byte formatting.

No.	Time	Source port	Destination port	Protocol	Length	Info
111	11.483334600	45992	36421	E2AP	194	RICindication RAN Function: 142
112	11.483341406	36421	45992	SCTP	62	SACK (Ack=3, Arwnd=62499870)
113	11.483380287	45992	36421	E2AP	102	RICsubscriptionResponse
114	11.483400549	45992	36421	E2AP	254	RICindication 143
115	11.483404234	36421	45992	SCTP	62	SACK (Ack=5, Arwnd=62499773)
116	11.488332436	45992	36421	E2AP	194	RICindication 142
117	11.488376822	45992	36421	E2AP	190	RICindication 144
118	11.488382500	36421	45992	SCTP	62	SACK (Ack=7, Arwnd=62499517)
119	11.488398468	45992	36421	E2AP	254	RICindication 143
120	11.491418129	38122	8094	UDP	313	38122 → 8094 Len=271 mac_stats
121	11.491431609	38122	8094	ICMP	341	Destination unreachable (Port unreachable)
122	11.492088021	38122	8094	UDP	598	38122 → 8094 Len=556 rlc stats
123	11.492096524	38122	8094	ICMP	590	Destination unreachable (Port unreachable)
124	11.492125237	38122	8094	UDP	313	38122 → 8094 Len=271 mac stats
125	11.492131101	38122	8094	ICMP	341	Destination unreachable (Port unreachable)
126	11.492224422	38122	8094	UDP	317	38122 → 8094 Len=275 pdcp stats
127	11.492229644	38122	8094	ICMP	345	Destination unreachable (Port unreachable)
128	11.492255723	38122	8094	UDP	598	38122 → 8094 Len=556 rlc stats
129	11.492259294	38122	8094	ICMP	590	Destination unreachable (Port unreachable)

Figure 5.16: RIC and iApp Indication sequence

The UDP packet contains the stats in text form as the data contents. The ICMP packet is quite strange as it informs the port 8094 that itself is unreachable. It also contains the raw text stats.



(a) iApp UDP Indication **(b)** iApp ICMP Destination Unreachable

Figure 5.17: iApp Indication packets details

5.2.2 Available xApps

The xApps provided by MOSAIC5G at the workshop have a very simple functionality. They simply subscribe to the RAN functions and then either display the incoming reports or use the API function `get_attr()` to inquire some specific attributes. Nonetheless it is interesting to monitor these statistics in real-time while performing Ping or Iperf tests to see how the values change.

The monitoring.py xApp makes use of the API to retrieve information as the number and RNTI of connected UEs and the layer's stats. As you can see, the stats show three different values because the API request ask for a window of 3 reports to be shown.

The other three have a similar functionality by just displaying 1 every N incoming reports, and some parameters require more explanation. In the rlc.py and pdcp.py stats there are many 0 values corresponding to stats not yet implemented, stats that have to do with RLC AM mode and PDCP out of order or dropped packets that with RFSIM do not occur. Besides, the RLC mode is set to 1 (Unacknowledged mode UM) contradicting the RLC mode shown in Fig. 5.10. Furthermore, it shows a PDCP mode that the standard does not describe. I suppose it is an error caused by copying the variables from the RLC layer.

```
                                        - monitoring.py -
COMM : INFO : CONNECTED TO RIC
XAPP : INFO : No ue connected
XAPP : INFO : Number of connected UEs: 1
XAPP : INFO : RNTI of connected UEs: [4660]
XAPP : INFO : Number of stored pdcp msg received from the 1st UE: 1000
XAPP : INFO : Number of stored rlc msg received from the 1st UE: 1000
XAPP : INFO : Number of stored mac msg received from the 1st UE: 1000
XAPP : INFO : pdcp: [[{'pdcp.rnti': [4660, 4660, 4660], 'pdcp.tstamp':
    [1661427633155597, 1661427633150598, 1661427633145598], 'pdcp.txpdu_bytes': [528538,
     528538, 528538], 'pdcp.rxpdu_bytes': [528488, 528488, 528488]}]]
XAPP : INFO : rlc: [[{'rlc.rnti': [4660, 4660, 4660], 'rlc.tstamp': [1661427633155613,
    1661427633150620, 1661427633145614], 'rlc.txpdu_bytes': [749104, 749104, 749104], '
    rlc.txpdu_retx_pkts': [0, 0, 0]}]]
XAPP : INFO : mac: [[{'mac.rnti': [4660, 4660, 4660], 'mac.tstamp': [1661427633155546,
    1661427633150547, 1661427633145547], 'mac.dl_aggr_tbs': [12734804, 12734804,
    12734804], 'mac.ul_aggr_tbs': [12727310, 12727310, 12727310]}]]
XAPP : INFO : Number of Mac msg received recv so far: 2000
XAPP : INFO : Number of RLC msg received recv so far : 2000
XAPP : INFO : Number of PDCP msg received recv so far : 2000
                                        - mac_cb.py -
XAPP : INFO  {
    "dl_aggr_bytes_sdus": 0,
    "dl_aggr_prb": 306050,
    "dl_aggr_retx_prb": 0,
    "dl_aggr_sdus": 6121,
    "dl_aggr_tbs": 7645129,
    "dl_mcs1": 9,
    "dl_mcs2": 0,
    "phr": 0,
    "pucch_snr": 62,
    "pusch_snr": 51,
    "rnti": 4660,
    "tstamp": 1661427547920547,
    "ul_aggr_bytes_sdus": 0,
    "ul_aggr_prb": 306100,
    "ul_aggr_sdus": 4,
    "ul_aggr_tbs": 7637635,
    "ul_mcs1": 0,
    "ul_mcs2": 0,
    "wb_cqi": 0
```

```
                                        - rlc_cb.py -
XAPP : INFO  {
    "mode": 1,
    "rbid": 1,
    "rnti": 4660,
    "rxbuf_occ_bytes": 0,
    "rxbuf_occ_pkts": 0,
    "rxpdu_bytes": 3028613,
    "rxpdu_dd_bytes": 0,
    "rxpdu_dd_pkts": 0,
    "rxpdu_dup_bytes": 0,
    "rxpdu_dup_pkts": 0,
    "rxpdu_ow_bytes": 0,
    "rxpdu_ow_pkts": 0,
    "rxpdu_pkts": 5180,
    "rxpdu_status_bytes": 0,
    "rxpdu_status_pkts": 0,
    "rxsdu_bytes": 807081932,
    "rxsdu_dd_bytes": 697897500,
    "rxsdu_dd_pkts": 465265,
    "rxsdu_pkts": 538069,
    "tstamp": 1661427864265616,
    "txbuf_occ_bytes": 0,
    "txbuf_occ_pkts": 0,
    "txpdu_bytes": 13577390,
    "txpdu_dd_bytes": 0,
    "txpdu_dd_pkts": 0,
    "txpdu_pkts": 14086,
    "txpdu_retx_bytes": 0,
    "txpdu_retx_pkts": 0,
    "txpdu_segmented": 15384,
    "txpdu_status_bytes": 0,
    "txpdu_status_pkts": 0,
    "txpdu_wt_ms": 0,
    "txsdu_bytes": 3014370,
    "txsdu_pkts": 2819
}
                                        - pdcp_cb.py -
XAPP : INFO  {
    "mode": 0,
    "rbid": 1,
    "rnti": 4660,
    "rxpdu_bytes": 3014370,
    "rxpdu_dd_bytes": 0,
    "rxpdu_dd_pkts": 0,
    "rxpdu_oo_bytes": 0,
    "rxpdu_oo_pkts": 0,
    "rxpdu_pkts": 2819,
    "rxpdu_ro_count": 0,
    "rxpdu_sn": 2819,
    "rxsdu_bytes": 405274316,
    "rxsdu_pkts": 270558,
    "tstamp": 1661427791040599,
    "txpdu_bytes": 405815432,
    "txpdu_pkts": 270558,
    "txpdu_sn": 221,
    "txsdu_bytes": 3008684,
    "txsdu_pkts": 2818
}
```

5.2.2.1 Wireshark packet sniffing

The FlexRIC's E42AP interface can be inspected with Wireshark as seen in Fig. 5.18, to view the xApp's communication.

No.	Time	Source port	Destination port	Protocol	Length	Info
18719	20.270933321	34674	9992	TCP	74	34674 → 9992 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1775509140 TSecr=0 WS=1024
18720	20.270941450	9992	34674	TCP	74	9992 → 34674 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1775509140 TSecr=1775509140 WS=1024
18721	20.270947974	34674	9992	TCP	66	34674 → 9992 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1775509140 TSecr=1775509140
18722	20.270971142	34676	9992	TCP	74	34676 → 9992 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1775509140 TSecr=0 WS=1024
18723	20.270974909	9992	34676	TCP	74	9992 → 34676 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1775509140 TSecr=1775509140 WS=1024
18724	20.270978846	34676	9992	TCP	66	34676 → 9992 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1775509140 TSecr=1775509140
18725	20.271045590	34678	9992	TCP	74	34678 → 9992 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1775509140 TSecr=0 WS=1024
18726	20.271049661	9992	34678	TCP	74	9992 → 34678 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1775509140 TSecr=1775509140 WS=1024
18727	20.271053908	34678	9992	TCP	66	34678 → 9992 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1775509140 TSecr=1775509140
18728	20.271082746	40222	9990	TCP	74	40222 → 9990 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1775509140 TSecr=0 WS=1024
18729	20.271089992	9990	40222	TCP	74	9990 → 40222 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1775509140 TSecr=1775509140 WS=1024
18730	20.271095919	40222	9990	TCP	66	40222 → 9990 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1775509140 TSecr=1775509140
18731	20.271336437	34674	9992	HTTP	262	GET HTTP/1.1
18732	20.271342713	9992	34674	TCP	66	9992 → 34674 [ACK] Seq=1 Ack=197 Win=65536 Len=0 TSval=1775509140 TSecr=1775509140
18733	20.271428760	40222	9990	HTTP	262	GET HTTP/1.1
18734	20.271433869	9990	40222	TCP	66	9990 → 40222 [ACK] Seq=1 Ack=197 Win=65536 Len=0 TSval=1775509140 TSecr=1775509140
18735	20.271467090	34676	9992	HTTP	262	GET HTTP/1.1
18736	20.271472863	9992	34676	TCP	66	9992 → 34676 [ACK] Seq=1 Ack=197 Win=65536 Len=0 TSval=1775509140 TSecr=1775509140
18737	20.271485256	34678	9992	HTTP	262	GET HTTP/1.1
18738	20.271489753	9992	34678	TCP	66	9992 → 34678 [ACK] Seq=1 Ack=197 Win=65536 Len=0 TSval=1775509140 TSecr=1775509140
18739	20.271554888	9990	40222	HTTP	239	HTTP/1.1 101 Switching Protocols
18740	20.271557611	40222	9990	TCP	66	40222 → 9990 [ACK] Seq=197 Ack=174 Win=65536 Len=0 TSval=1775509140 TSecr=1775509140
18741	20.271583406	9992	34678	HTTP	239	HTTP/1.1 101 Switching Protocols
18742	20.271586584	34678	9992	TCP	66	34678 → 9992 [ACK] Seq=197 Ack=174 Win=65536 Len=0 TSval=1775509140 TSecr=1775509140
18743	20.271643525	40222	9990	WebSocket	81	WebSocket Binary [FIN] [MASKED] INIT
18744	20.271722368	9992	34674	HTTP	239	HTTP/1.1 101 Switching Protocols
18745	20.271725078	34674	9992	TCP	66	34674 → 9992 [ACK] Seq=197 Ack=174 Win=65536 Len=0 TSval=1775509141 TSecr=1775509141
18746	20.271788765	9992	34676	HTTP	239	HTTP/1.1 101 Switching Protocols
18747	20.271791453	34676	9992	TCP	66	34676 → 9992 [ACK] Seq=197 Ack=174 Win=65536 Len=0 TSval=1775509141 TSecr=1775509141
18748	20.271792149	9990	40222	WebSocket	74	WebSocket Binary [FIN]
18749	20.272375861	40222	9990	WebSocket	86	WebSocket Binary [FIN] [MASKED] ID=1; PING
18750	20.272440400	47812	9991	TCP	74	47812 → 9991 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1775509141 TSecr=0 WS=1024
18751	20.272448517	9991	47812	TCP	74	9991 → 47812 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1775509141 TSecr=1775509141 WS=1024
18752	20.272463985	47812	9991	TCP	66	47812 → 9991 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1775509141 TSecr=1775509141
18753	20.272468220	9990	40222	WebSocket	77	WebSocket Binary [FIN] PONG
18754	20.272585945	47812	9991	HTTP	262	GET HTTP/1.1
18755	20.272590380	9991	47812	TCP	66	9991 → 47812 [ACK] Seq=1 Ack=197 Win=65536 Len=0 TSval=1775509141 TSecr=1775509141
18756	20.272632674	9991	47812	HTTP	239	HTTP/1.1 101 Switching Protocols
18757	20.272636811	47812	9991	TCP	66	47812 → 9991 [ACK] Seq=197 Ack=174 Win=65536 Len=0 TSval=1775509141 TSecr=1775509141
18758	20.272775657	47812	9991	WebSocket	112	WebSocket Binary [FIN] [MASKED] ID=1;REQUEST; REPORT=MAC;INTERVAL=10
18759	20.272843077	9991	47812	WebSocket	82	WebSocket Binary [FIN] ANSWER_OK
18760	20.273206355	47812	9991	WebSocket	112	WebSocket Binary [FIN] [MASKED] ID=1;REQUEST;REPORT=RLC;INTERVAL=10
18761	20.273297078	9991	47812	WebSocket	82	WebSocket Binary [FIN] ANSWER_OK
18762	20.273382736	45992	36421	E2AP	194	RICindication RAN Function: 142
18763	20.273446533	45992	36421	E2AP	190	RICindication RAN Function: 144
18764	20.273458502	36421	45992	SCTP	62	SACK (Ack=5273, Arwnd=62499874)
18765	20.273493102	38122	8094	UDP	340	38122 → 8094 Len=298 mac_stats: ...
18766	20.273512614	38122	8094	ICMP	368	Destination unreachable (Port unreachable)
18767	20.273537294	9992	34678	WebSocket	369	WebSocket Binary [FIN] mac_stats: ...
18768	20.273542204	34678	9992	TCP	66	34678 → 9992 [ACK] Seq=197 Ack=477 Win=65536 Len=0 TSval=1775509142 TSecr=1775509142
18769	20.273555166	45992	36421	E2AP	254	RICindication RAN Function: 143
18770	20.273557462	9992	34674	WebSocket	369	WebSocket Binary [FIN] mac_stats: ...
18771	20.273560292	34674	9992	TCP	66	34674 → 9992 [ACK] Seq=197 Ack=477 Win=65536 Len=0 TSval=1775509142 TSecr=1775509142
18772	20.273594391	9992	34676	WebSocket	369	WebSocket Binary [FIN] mac_stats: ...
18773	20.273598256	34676	9992	TCP	66	34676 → 9992 [ACK] Seq=197 Ack=477 Win=65536 Len=0 TSval=1775509142 TSecr=1775509142
18774	20.273649031	38122	8094	UDP	322	38122 → 8094 Len=280 pdcp_stats: ...
18775	20.273658464	38122	8094	ICMP	350	Destination unreachable (Port unreachable)
18776	20.273672335	9992	34678	WebSocket	351	WebSocket Binary [FIN] pdcp_stats: ...
18777	20.273674799	34678	9992	TCP	66	34678 → 9992 [ACK] Seq=197 Ack=762 Win=65536 Len=0 TSval=1775509143 TSecr=1775509143
18778	20.273740318	9992	34674	WebSocket	351	WebSocket Binary [FIN] pdcp_stats: ...
18779	20.273745941	34674	9992	TCP	66	34674 → 9992 [ACK] Seq=197 Ack=762 Win=65536 Len=0 TSval=1775509143 TSecr=1775509143
18780	20.273776603	9992	34676	WebSocket	351	WebSocket Binary [FIN] pdcp_stats: ...
18781	20.273780074	34676	9992	TCP	66	34676 → 9992 [ACK] Seq=197 Ack=762 Win=65536 Len=0 TSval=1775509143 TSecr=1775509143
18782	20.273813255	38122	8094	UDP	605	38122 → 8094 Len=563 rlc_stats: ...
18783	20.273821501	38122	8094	ICMP	590	Destination unreachable (Port unreachable)
18784	20.273836011	9992	34678	WebSocket	634	WebSocket Binary [FIN] rlc_stats: ...
18785	20.273838855	34678	9992	TCP	66	34678 → 9992 [ACK] Seq=197 Ack=1330 Win=65536 Len=0 TSval=1775509143 TSecr=1775509143
18786	20.273848845	9992	34674	WebSocket	634	WebSocket Binary [FIN] rlc_stats: ...
18787	20.273851020	34674	9992	TCP	66	34674 → 9992 [ACK] Seq=197 Ack=1330 Win=65536 Len=0 TSval=1775509143 TSecr=1775509143
18788	20.273882657	9992	34676	WebSocket	634	WebSocket Binary [FIN] rlc_stats: ...
18789	20.273885271	34676	9992	TCP	66	34676 → 9992 [ACK] Seq=197 Ack=1330 Win=65536 Len=0 TSval=1775509143 TSecr=1775509143
18790	20.276557886	47812	9991	WebSocket	113	WebSocket Binary [FIN] [MASKED] ID=1;REQUEST;REPORT=PDCP;INTERVAL=10
18791	20.276651923	9991	47812	WebSocket	82	WebSocket Binary [FIN] ANSWER_OK
18792	20.278357690	45992	36421	E2AP	194	RICindication

Figure 5.18: xApp communication over the E42AP interface

The Wireshark capture of the iApp ↔ xApp (`monitoring.py`) communication over the E42AP interface begins with the TCP establishment handshake and HTTP protocol INIT to switch to the WebSocket protocol with which E42AP is implemented.
Then, the iApp ping address port `9990` is used by the xApp to initiate the communication and ping the iApp indicating its ID. Over the iApp message address port `9991`, the xApp requests the type of report and reporting interval indicating the ID previously told for each SM (RAN Function) it is interested in subscribing to.
Finally, once the E2 Node sends the RICIndication message and it is forwarded to the SM iApp, the iApp uses its pub/sub address port `9992` to report the requested stats. For some unknown reason, the reports are sent after the INIT and PING even if the xApp does not subscribe to them. This happens in several runs.

All of this packet exchange over E42AP is logged by the RIC as well. The following log shows three consecutive xApps connecting to the RIC.

```
[E2AP] SUBSCRIPTION RESPONSE received
[E2AP] SUBSCRIPTION RESPONSE received
[iApp]: Server ping started
[iApp]: Server Request/Reply started
[E2AP] SUBSCRIPTION RESPONSE received
Value of the buffer in parse_xapp = INIT
NODE0: RECEIVED INIT
Before generate id
generate id called
After generate id
NODE0: SENDING REQUEST_ID 1
Value of the buffer in parse_xapp = ID=1;PING
NODE0: RECEIVED PING
Value of the buffer in parse_xapp = ID=1;PING
NODE0: RECEIVED PING
[iApp]: Disconnected xApps = 1
[iApp]: Deleted xApp id = 1
Value of the buffer in parse_xapp = INIT
NODE0: RECEIVED INIT
Before generate id
generate id called
After generate id
NODE0: SENDING REQUEST_ID 2
Value of the buffer in parse_xapp = ID=2;PING
NODE0: RECEIVED PING
Value of the buffer in parse_xapp = ID=2;PING
NODE0: RECEIVED PING
[iApp]: Disconnected xApps = 1
[iApp]: Deleted xApp id = 2
Value of the buffer in parse_xapp = INIT
NODE0: RECEIVED INIT
Before generate id
generate id called
After generate id
NODE0: SENDING REQUEST_ID 3
Value of the buffer in parse_xapp = ID=3;PING
NODE0: RECEIVED PING
Value of the buffer in parse_xapp = ID=3;REQUEST;REPORT=MAC;INTERVAL=10;
NODE0: RECEIVED REQUEST
Value of the buffer in parse_xapp = ID=3;REQUEST;REPORT=RLC;INTERVAL=10;
NODE0: RECEIVED REQUEST
```

5.2.3 Throughput xApp

The `matplotlib` library suggested to visualize the data is not best suited for this use case for several reasons. On the one side it lacks multi threading environment support, as it is not thread-safe, and most GUI backends require being run from the main thread which, with this implementation of xApps, was not feasible. [31] On the other side, while having a GUI to show the data is a nice feature to have, it is unsuitable for remote command console access over ssh, which is a common use case. In fact, the UHD includes a utility to check the correct USRP function by showing the radio spectrum and, as it is meant to be run in a console environment and over ssh, it uses text characters to graph the spectrum.

Following this idea, I propose another library to graph with text in the console: *Plotille*. This open-source library allows "to visualize plots, scatter plots, histograms and heatmaps in the terminal using braille dots, and foreground and background colors - with (almost) no dependencies." [32] It is designed around the `Figure` class, used to indicate the plot type and modify the plot's visual appearance before printing it to the console as any other print to terminal.

The throughput xApp is based on the `get_attr` function of the xApp SDK's API, in a similar way to the monitoring xApp. This function returns a list of attributes that can be filtered by UE, attribute and amount of past messages.

Even though the challenges' objective is to compute throughput per UE or Base Station, with the current setup with `phy-test` mode only one gNB and UE can be connected and so, even if the implementation would allow to distinguish, we would not be able to see it. Therefore, this throughput xApp design computes total throughput for the gNB and UE.

Regarding the throughput calculation, it is done by computing the difference in data bytes over a timestamp delta increment, for each pre-defined period in the variable `period_to_run_logic`. The data unit used for each layer are the following stats: `mac.dl_aggr_tbs`, `mac.ul_aggr_tbs`, `rlc.txpdu_bytes`, `rlc.rxpdu_bytes`, `pdcp.txpdu_bytes` and `pdcp.rxpdu_bytes`.

As this API request and throughput calculation needs to be done periodically, the logic must be defined inside the `run_periodic` function and the `Xapp` class object's variable `trigger_periodic_func` set to `True`.

For the xApp to be able to call the API function it must subscribe to the report of the SMs, although no message handler is needed as it does not rely in the periodic reports to get the stats. Another logic could be designed to do so, instead of using the API function.

In order to hold the requested and calculated data, several variables have to be declared. For each incoming timestamp and data stat a list is created. The data lists are appended an initial value of 0 to allow the computing logic to work for the first period. The calculated time deltas and throughputs the `deque` variable type (*Doubly Ended Queue* from the `collections` module) is used so that the graph can show the last a fixed number of periods without shrinking over long runs. Finally, a flag to mark the first time the logic is executed is set to 1.

For the terminal console graph itself, the `Plotille Figure` object is created and labeled. For aesthetics purposes, the y axis' lower limit is zero-ed. The x axis limits cannot be set because for long runs only a set amount of periods will be shown and will not start in 0 but in a different time delta each period. Furthermore, the graph's height and width are adjusted to fit the terminal window size and the axis ticks formatted to three decimal places.

Regarding the logic itself, it is repeated equally for each layer and transmission direction. It starts with the API attributes request and appended to the variables by selecting the appropriate list values returned by the `get_attr` function. Then the time deltas are computed as the difference of the actual and prior values in the timestamp list. The first period will return a delta of 0 as only one element is in the list and it is subtracted from itself. The timestamps provided by the RIC are microsecond based (16-digit) so the deltas are converted to seconds.

The first time the logic is executed the throughput cannot be computed as only one absolute bytes stat is provided and the change from 0 to a high absolute value results in a spike in throughput. To solve this, the first period appends the bytes stat to the The following times the throughput can be computed as the difference of the last two bytes stats over the time delta increment and have the figure plot printed.

The throughput xApp python code can be found at Appendix C.

Thanks to this xApp there is now a much clearer and graphical way to view the data flow within each layer. With it is possible to analyze each layer's throughput when idle in `phy-test` mode and when performing a ping or Iperf benchmark.

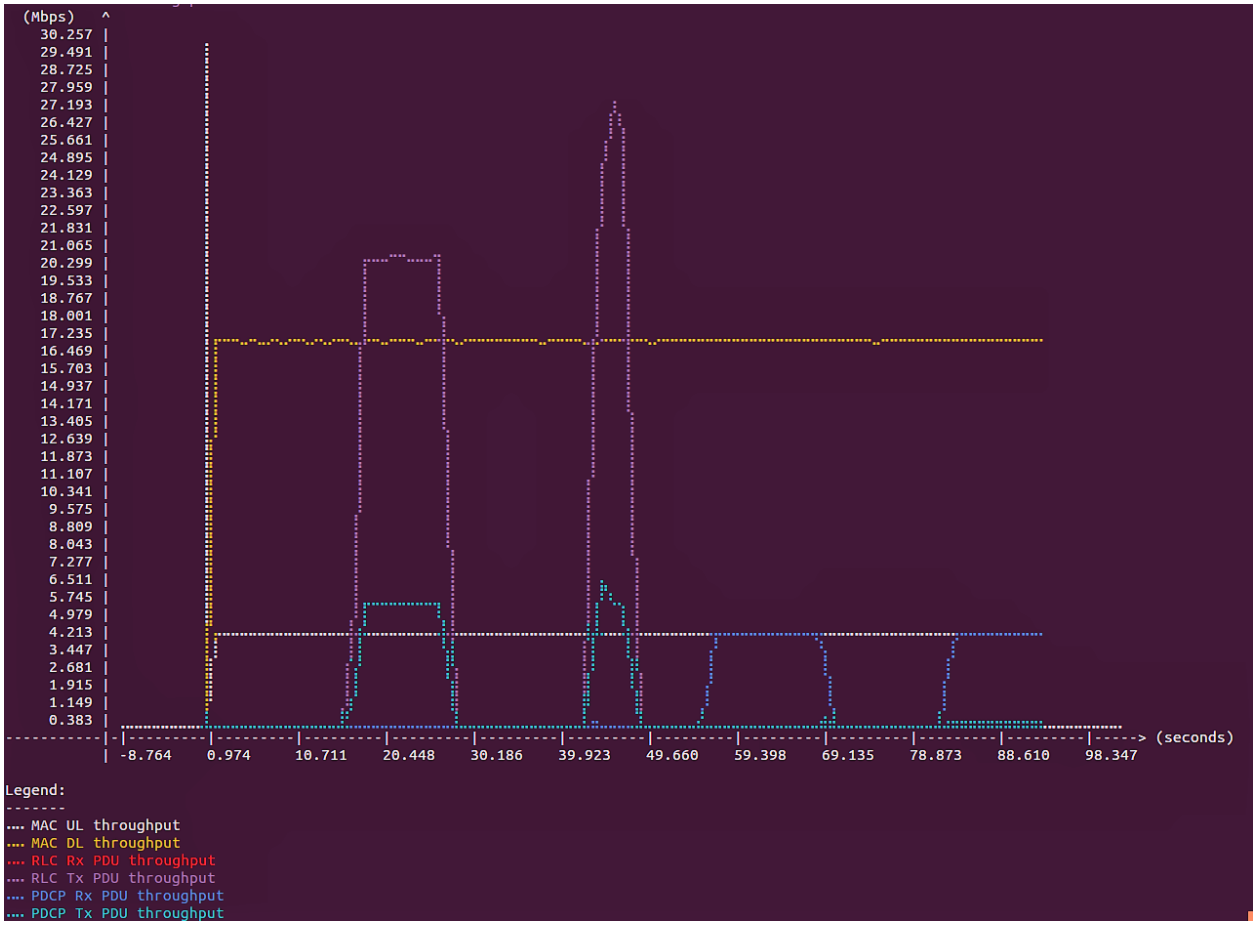


Figure 5.19: All layers' throughput xApp showing different Iperf runs.

As mentioned previously, in `phy-test` mode random UL and DL traffic is generated at every

scheduling opportunity by both gNB and UE. This scheduling is done by the MAC layer. For this reason, the MAC throughput is constant when idling (i.e. not sending traffic over higher layers). The MAC constant throughput value depends on the amount of PRBs, MCS, number of layers and slot allocation for both UL and DL configurable with the `phy-test` option. In Fig. 5.19 these parameters are set differently for UL and DL as an example.

The RLC and PDCP layers show no throughput when idle, and an increase when performing ping or Iperf tests.

When performing an Iperf benchmark over UDP with high bandwidth (-b option) as in Fig. 5.20, the Tx PDCP layer shows an overly high DL throughput that does not totally flow to the RLC layer. This is because it is an unacknowledged mode, and so Iperf doesn't wait and sends all packets during 10s. The RLC layer then buffers all the incoming stream and sends it a the rate it can until the buffer empties

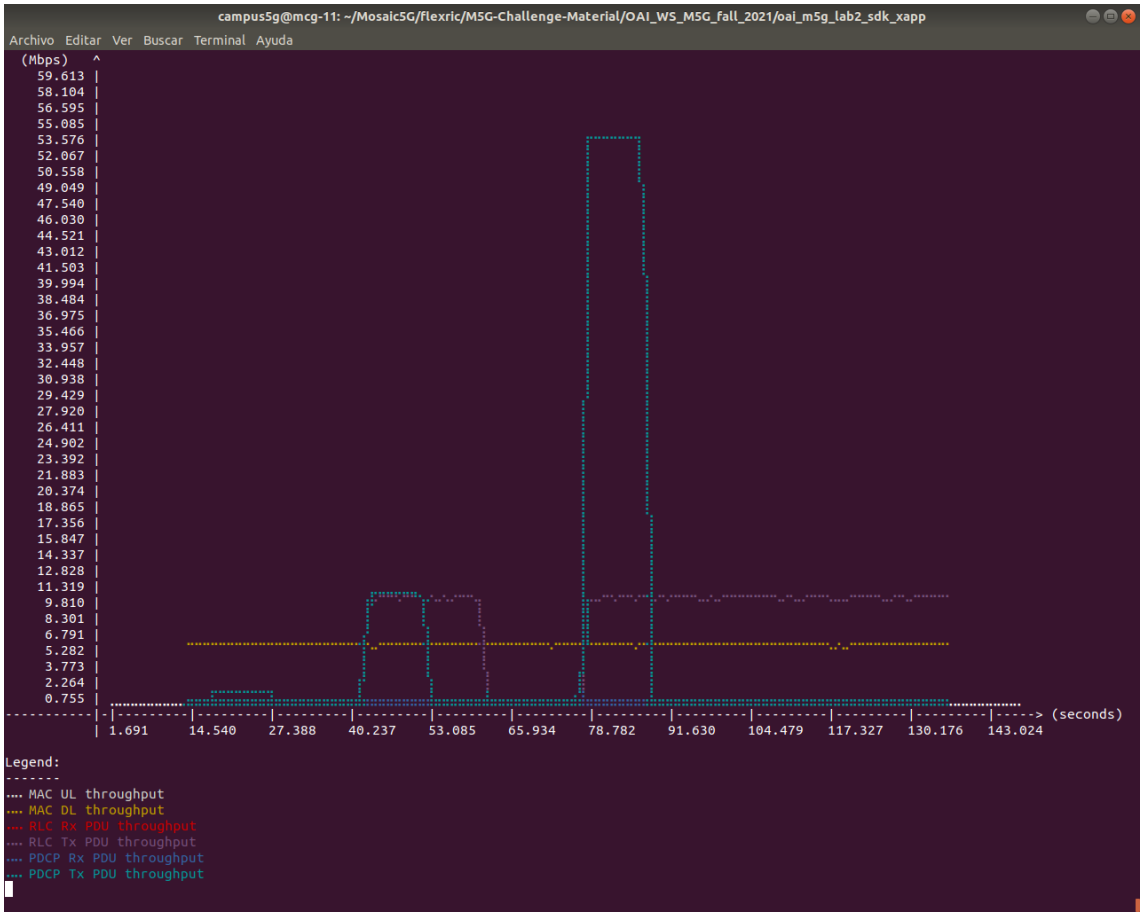


Figure 5.20: Throughput xApp showing three DL UDP iperf test with 1M, 10M and 50M of bandwidth.

When performing the test over TCP as in Fig. 5.21, the PDCP layer's throughput is limited and doesn't cause the RLC layer to buffer. Therefore, the RLC and PDCP layers show traffic during the 10 seconds that the test lasts for.

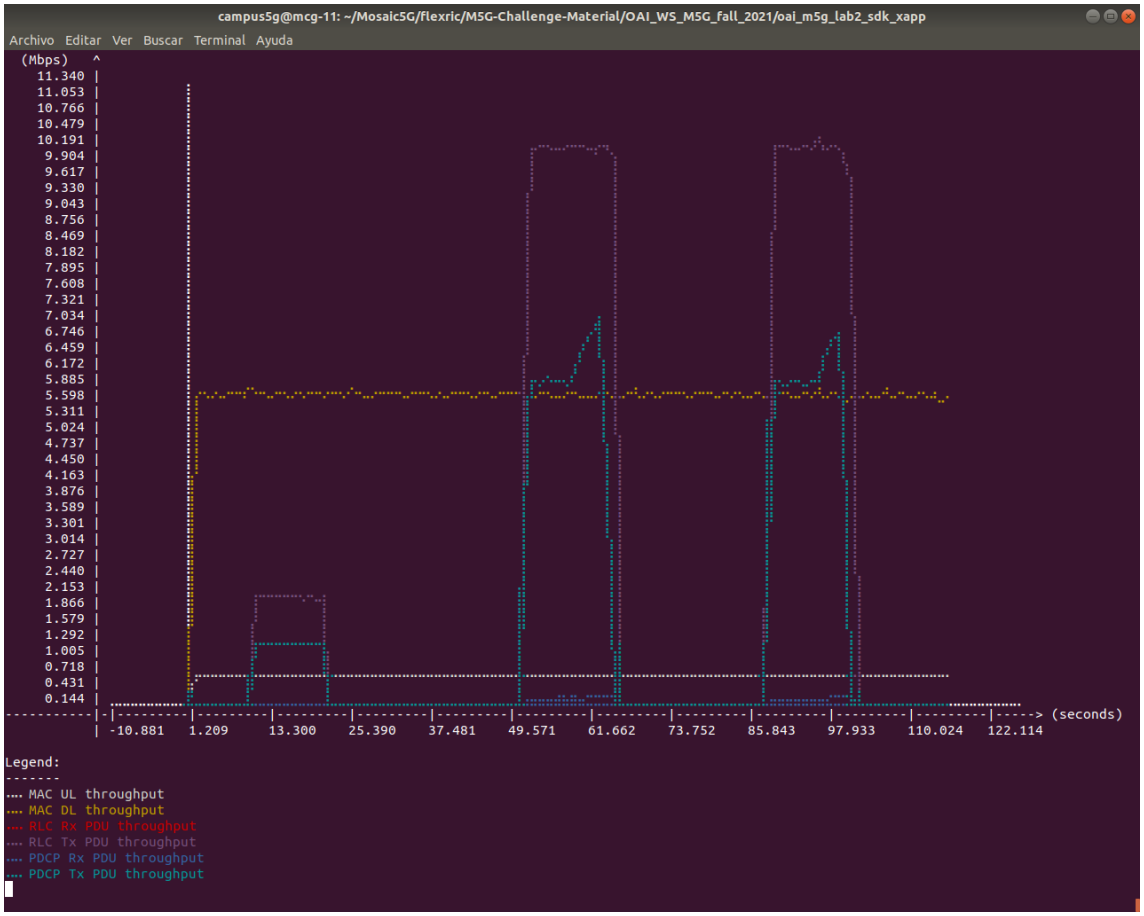


Figure 5.21: Throughput xApp showing three DL TCP iperf test with 1M, 10M and 50M of bandwidth.

In reception, the RLC and PDCP throughput is limited to the MAC's throughput. It is interesting to see in Fig. 5.22 how in TCP the TX throughput show the ACKs while in UDP it does not.

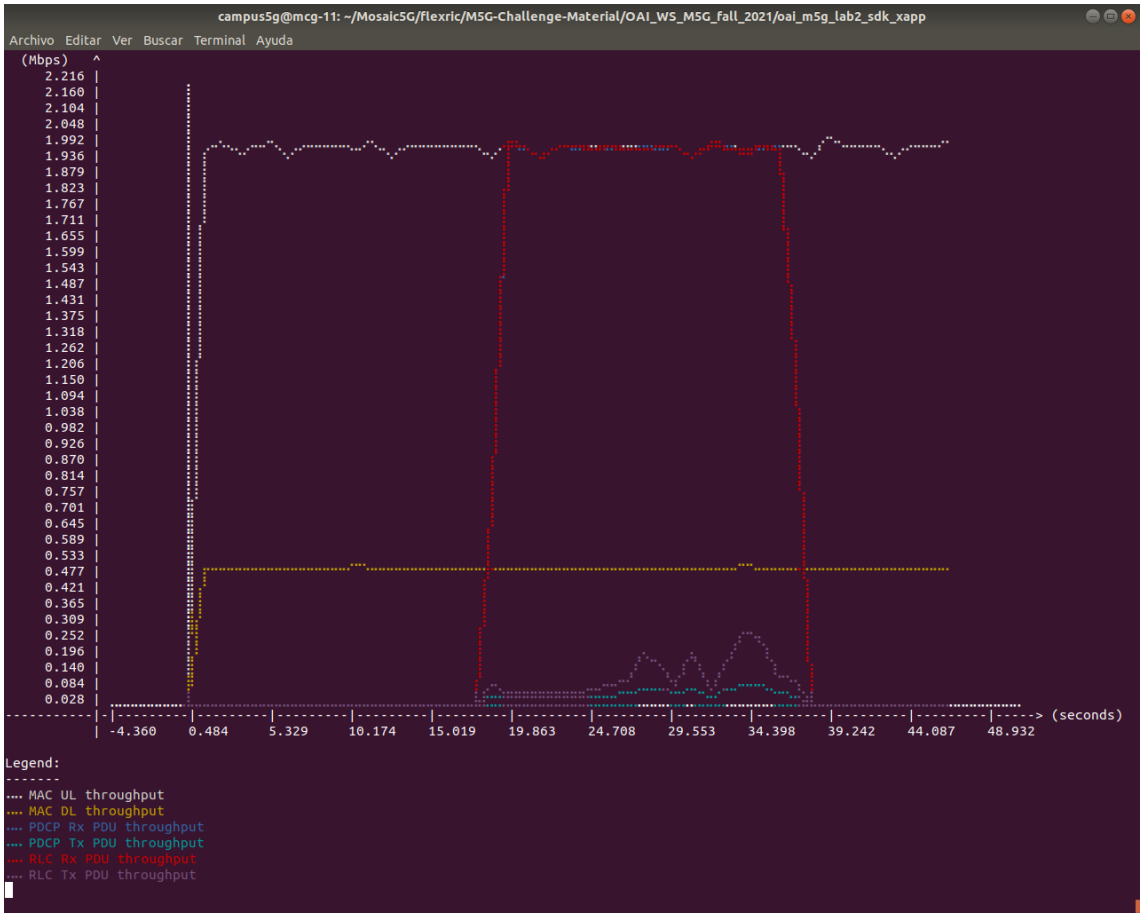


Figure 5.22: Throughput xApp showing a UL TCP iperf test.

5.2.4 O-RAN RIC connection

Having changed the RIC's IP address and port to match O-RAN's and having FlexRIC and OAI rebuild, the gNB can be executed. Its E2 Agent will now connect to O-RAN's RIC.

At this moment, the e2mgr's log at the E2Setup procedure shows the incoming E2 data from the gNB indicating its plmn-id and available RAN Functions, as well as the connectivity status, among other messages.


```
{"crit":"INFO","ts":1659089494033,"id":"E2Manager","msg":"#RnibDataService.SaveNodeb
    nodebInfo: ran_name:\"gnb_208_099_0000e000\" global_nb_id:{plmn_id:\"02F899\" nb_id
    :\"00000000000000000111000000000\"} node_type:GNB gnb:{ran_functions:{ran_function_id
    :142 ran_function_definition:\"4D41435F53544154535F5630\"} ran_functions:{
    ran_function_id:143 ran_function_definition:\"524C435F53544154535F5630\"}
    ran_functions:{ran_function_id:144 ran_function_definition:\"504443505
    F53544154535F5630\"} ran_functions:{ran_function_id:145 ran_function_definition
    :\"534C4943455F53544154535F5630\"} gnb_type:GNB} associated_e2t_instance_address
    :\"10.104.232.64:38000\" setup_from_network:true","mdc":{"time":"2022-07-29
    10:11:34.033"}}

{"crit":"INFO","ts":1659089494072,"id":"E2Manager","msg":"#RanConnectStatusChangeManager
    .ChangeStatus RAN name: gnb_208_099_0000e000, updating RanListManager... status:
    CONNECTED","mdc":{"time":"2022-07-29 10:11:34.072"}}
```


The connection status and identification for the E2 Agent can also be inquired via an cURL API request to the e2mgr.


```
$ curl -X GET http://10.103.66.181:3800/v1/nodeb/states | jq .  - e2mgr -
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   439  100   439    0     0   428k      0 --:--:-- --:--:-- --:--:--  428k
[
  {
    "inventoryName": "gnb_010_015_000002a0",
    "globalNbId": {
      "plmnId": "10F051",
      "nbId": "00000000000000000000000101010"
    },
    "connectionStatus": "DISCONNECTED"       - Previous gNB connection -
  },
  {
    "inventoryName": "gnb_208_099_0000e000",
    "globalNbId": {
      "plmnId": "02F899",
      "nbId": "00000000000000000111000000000"
    },
    "connectionStatus": "CONNECTED"          - Actual gNB connection -
  }
```

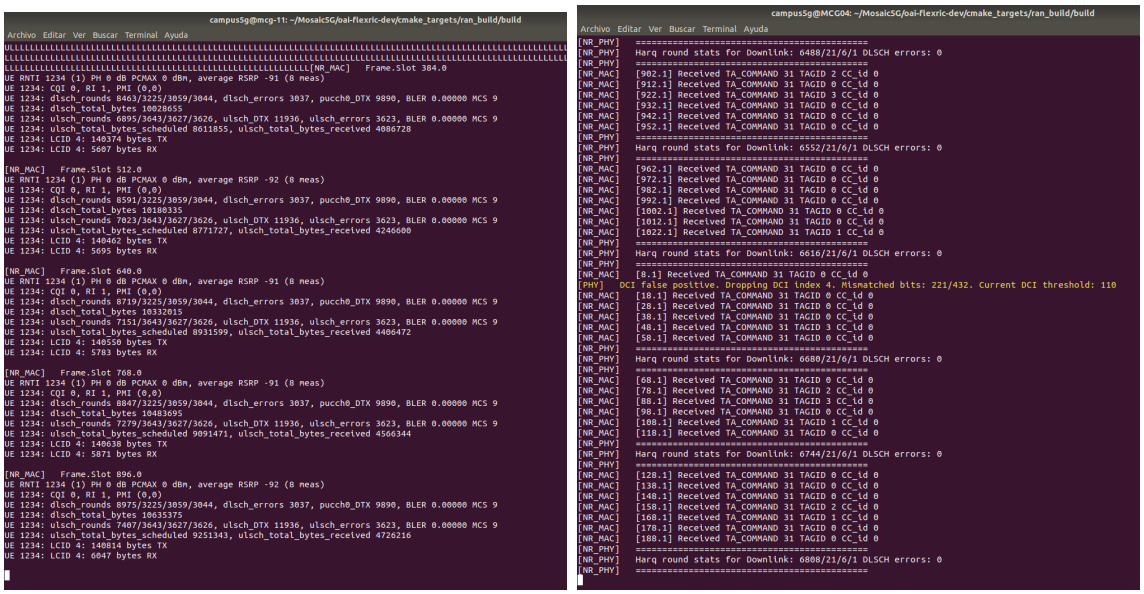

The communication between the E2 Agent and O-RAN RIC over the E2 Interface can be also seen in Wireshark. It is not shown because it is very similar to that of Fig. 5.12

5.3 OAI RAN New Version

FlexRIC's first version under `master` branch integrates with OAI tag `integration_2021_wk51_a` from December 2021, while FlexRIC's most recent version under`dev` branch integrates with the actively updated branch `nr-mac-rlc-pdcp-stats`. At explore the possibilities of using true radio heads in the deployment, this new OAI version was installed, although FlexRIC's `dev` version was not ready yet and could not be checked out.

This new version solves the issue with the USRPs that the OAI's older version had, and so it can be run without the RF Simulator, still in `phy-test` mode. The config file used for the execution is the one provided in `/ci-scripts/conf_files/gnb.band78.sa.fr1.106PRB.usrp310.conf`.

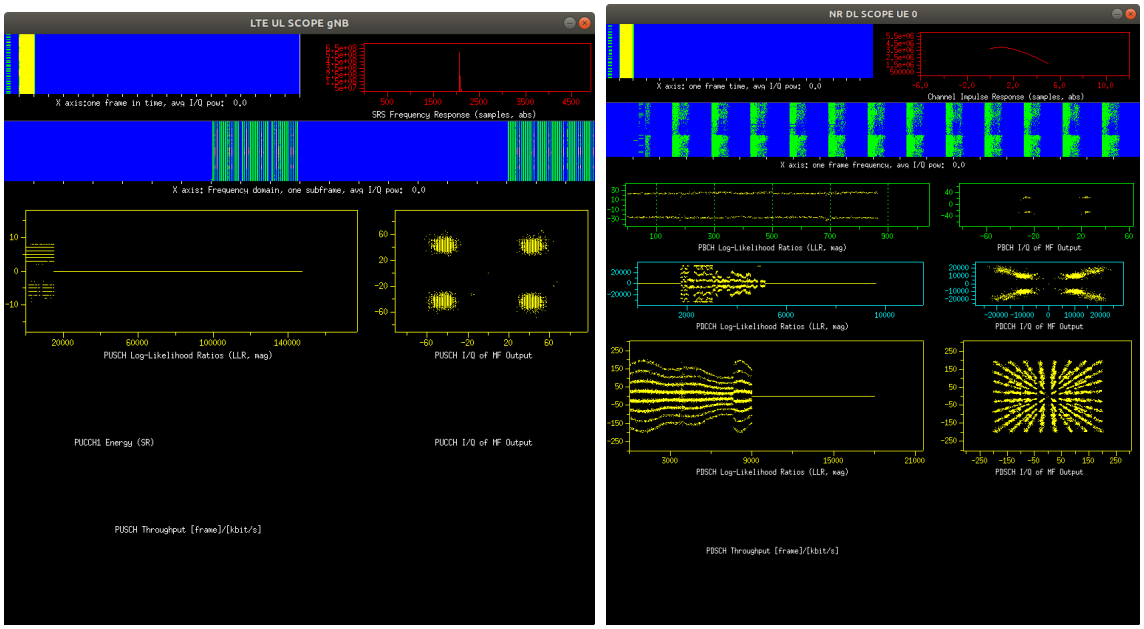
In this version and using USRPs the gNB and UE logs in Fig. 5.23 show as it can send and receive bytes to LCID 4, the `ulscg_total_bytes_received` counter increases and the `ulsch_errors` and remain constant.



(a) gNB log **(b)** UE log

Figure 5.23: OAI RAN newer version's logs

The Scopes in Fig. 5.24 show a correct transmission over every channel. The gNB's PUSCH reception is correct, showing a proper log-likelihood ratio for a 4QAM constellation. The UE's PDSCH 64QAM constellation does update and shows a log-likelihood adequate for an over-the-air (OTA) channel. In general, in comparison with the previous version tested, the scope shows a good transmission and updates in real-time.



(a) gNB scope **(b)** UE scope

Figure 5.24: OAI RAN newer version's scopes

5.3.1 Throughput Benchmark

Having finally accomplished OTA transmission with the USRPs as radio heads, it is key to also benchmark the throughput available. These results will differ from the ones performed with the RF Simulator as it is now possible to run at real-time processing speeds, instead than at CPU capable speeds. As a result of this, there are some configurations in which the CPU will lack processing power, making the UE unstable and dropping USRP incoming frames, so no benchmark could be performed in those situations. Furthermore, real-time speed means that the RLC layer buffer can overflow if too much incoming data stream is injected. In these cases, the gNB will output many warnings as the following, indicating that the SDU buffer is full and the incoming SDU is rejected.

```
[RLC]  /home/campus5g/Mosaic5G/oai-FlexRIC-dev/openair2/LAYER2/nr_rlc/
    nr_rlc_entity_um.c:580:nr_rlc_entity_um_recv_sdu: warning: SDU rejected,
    SDU buffer full
```

This warning indicates that the RLC entity used corresponds to the UM mode, matching what the xApps showed in section 5.2.2 and contradicting the Wireshark packets in Fig. 5.10. Although this is a new OAI version, I do not thing the default RLC mode has changed and so, after not finding any code mismatch, I suspect the correct RLC mode is UM.

In other cases, as the RLC layer is in UM mode, if some packet is lost in transmission there is no way to fix this and the UE outputs the following warnings:

```
[RLC] openari2/layer2/nr_rlc/nr_rlc_entity_um.c:352 nr_rlc_entity_um_recv_pdu:
    warning: discard PDU, SN (1792) < rx_next_reassembly (1796)
```

```
 – or –
[PDCP] discard NR PDU rcvd_count = 1600, entity->rx_deliv 1605,sdu_in_list 0
```

DL throughput in Mbps						
	50 PRBs			106 PRBs		
# of slots	1	6	12	1	6	12
MCS 9	0.92 / 1.00	5.04 / 6.00	6.85 / 11.99	1.94 / 2.10	9.41 /12.60	—
MCS 16	1.79 /1.95	8.73 / 11.68	—	3.80 / 4.10	—	—
MCS 28	3.89 / 4.20	18.8 / 25.21	—	8.19 / 8.81	—	—
DL jitter in ms						
	50 PRBs			106 PRBs		
# of slots	1	6	12	1	6	12
MCS 9	12.585	2.258	1.407	6.061	1.660	—
MCS 16	6.375	1.286	—	3.381	—	—
MCS 28	3.191	0.831	—	1.779	—-	—

Table 5.2: Over-The-Air UDP throughput benchmark

Together with the Iperf throughput benchmark, ping test have been performed to measure latency at around 11.5 mili-seconds for every stable configuration. As with the RFSIM benchmark, in Fig. 5.25 a comparison between theoretical and measured throughput is shown. There are several points zero-ed to show that those configurations were unstable. The points that to show a throughput value appear closer to the theoretical line, indicating that the scenario with USRPs achieves a throughput closer to the maximum.

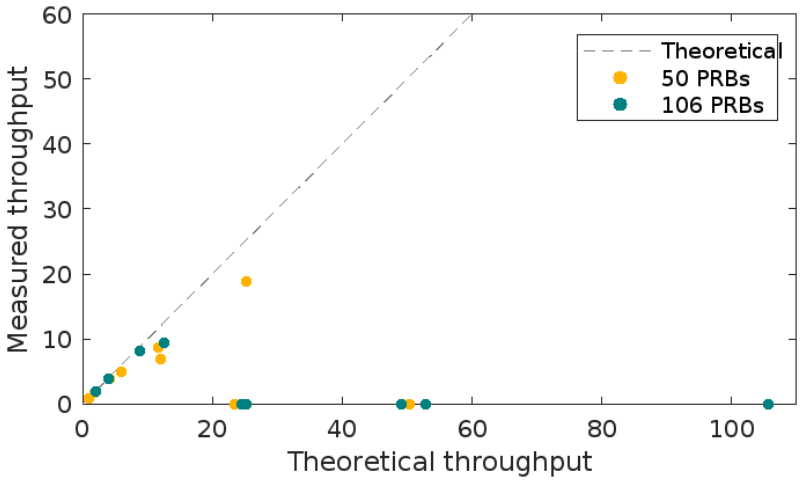


Figure 5.25: USRP theoretical throughput vs. measured throughput

It is possible to compare the throughput results from the RFSIM and USRP scenarios, as seen in Fig. 5.26, where lines of best fit have been added. For the RFSIM case, instead of a line is a second-degree polynomial curve because the best fit line did not cross the origin and this curve has a higher R squared value. It is clear that RFSIM is limited by processing power while with USRPs the throughput is close to theoretical.

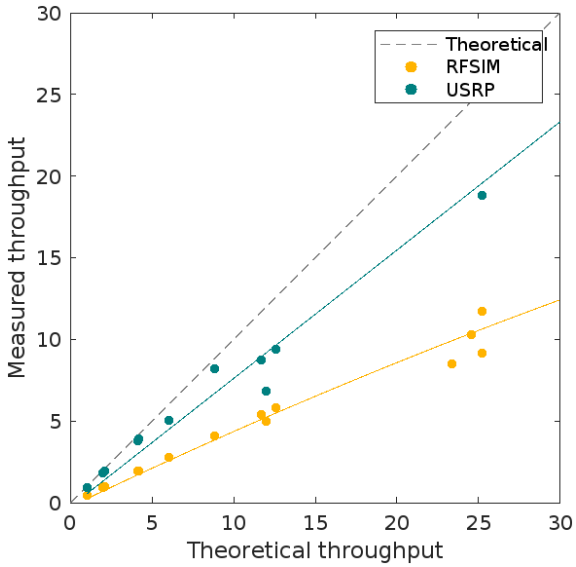


Figure 5.26: RFSIM vs USRP throughput

Chapter 6

Conclusions and Future Work

This chapter finally summarizes all the work done in this project, offering some conclusions, discussing the limitations encountered and providing a guideline for future works in this field.

6.1 Conclusions

This Bachelor's degree final project has focused on exploring the Open RAN novel concept and a recent implementation of the O-RAN standard called FlexRIC. These are the first steps towards open, intelligent, virtualized mobile networks that will become the norm in the coming years, and this project researched them. This study has consisted of several stages of development to acquire the specific knowledge and to make use of it, thanks to the abilities learned throughout my Telecommunication's Engineering degree.

The project has firstly consisted in a thorough review of the technical aspects that define the 5G NR and O-RAN standards, focusing on the Radio Access Network and E2 Interface. The software and hardware platforms used by the community to research and test mobile networks have also been examined.

Next, the appropriate equipment has been configured to achieve the setup to be used in the RAN deployment. In this section there has been a great deal of learning about the physical and operating system parameters involved. The setup has had plenty of troubleshooting work regarding the physical connections, computing power and dealing with USRP errors, until arriving at the final configuration explained in this report.

OpenAirInterface has many options, modes and tools available for a RAN deployment which had to be explored to learn what was implemented, their usage, what they can be used to and to select the ones relevant for this project. The same had to be done with the FlexRIC project, to conclude what was worth studying. Then, the different aspects that wanted to be tested and analyzed in-depth in the results had to be defined.

So, one part of this work has been to study the OAI RAN using the SA and `phy-test` modes. Although at first the connection had to be done via RFSIM, later on the USRPs could be used. The different modes have been analyzed using ROMES measurement software, an advanced spectrum analyzer, the OAI Scope, Wireshark and the provided logs. As a network analytic measurement, throughput benchmarks were performed on simulated and true radio heads and the results discussed

and compared between them and against the theoretical values.

The other part of this work has focused on has been the FlexRIC implementation of the O-RAN standard. The analysis conducted has mainly had to do with the near-RT RIC, the E2 Interface, the Service Models and available xApps. All has been examined both in code and in execution, using Wireshark to inspect the E2AP and E42AP message exchanges. As proposed by a Challenge, a Troughput xApp has been designed and showcased its usefulness in different UDP and TCP communication examples. Finally, as FlexRIC claims to be O-RAN compatible, a concise test to check that connection between the E2 Agent and O-RAN RIC has been performed.

Some key takeaways from this project are that, although a usual 5G deployment includes the Core, it has been interesting to test the RAN exclusively as very few reports are available and it is a great subject of study. OAI provides several options to run an isolated RAN that would not have otherwise been explored. This has obligated a better understanding of the physical layer and RAN protocols. In addition, it has been possible to verify that OAI achieves close to maximum throughputs when using USRPs, only limited by the CPU performance.

Moreover, FlexRIC introduces a great alternative to O-RAN's own implementation and it will be nice to see how they evolve into the future, with its different design decisions. It also provides a good framework for xApp developing that has been tried out with the custom made Throughput xApp featured in this work. The RLC mode discrepancy would still need to be resolved by the developers.

Finally, due to time constrains and the delay to release FlexRIC's new version, it could not be tested. It would have been interesting to test it with USRPs and to try the new features it brings.

6.2 Limitations

6.2.1 PC performance

A lot of work has been done in this project to deal with the PC's lack of enough performance power. In the beginning, the PC machines were selected as they were already in use in other deployments with USRP B210 in 4G LTE. Switching to newer and more capable USRP N310 and 5G NR resulted in an increase in the processing power needed. We must note that 5G NR introduces the use of higher bandwidths compared to 4G LTE as well as higher computational demand with the new LDPC encoding.

This has resulted in the need to extremely tweak the PC's performance configuration as shown in Section 3.1 to prevent Under-runs, Late processing and Dropped packet errors from UHD. Even another networking driver DPDK was tried out to replace Linux networking stack with a low overhead user-land based driver. This tweaking has taken up very valuable time of this project that could have been used to focus on more relevant aspects.

In the end, the PC's under-performance has limited the throughput benchmark in Section 5.3.1 as some configurations were unstable due to lack of CPU power.

In view of this, I would now have explored other possibilities such as employing more powerful computers or offloading the LDCP calculations to an FPGA or GPU board. To use another board, together with the PCIe SFP+ network card, the motherboards should have at least two PCIe x16 connectors, which the ones in the MCG's Lab do not have.

6.2.2 Server setup

As a side note, I have also tried to run a similar set up but virtualized by using Virtual Machines (VMs) inside a rack server instead of two different physical PCs. This is motivated by the new trend in network virtualization introduced with 5G, so that many of the network functions can be run on remote, virtualized, scalable environments.

There are several challenges regarding this setup, mainly with the USRPs high-bandwidth communications and processing, and the available resources at the MCG's Lab made it more convenient for us to use discrete physical machines for the Proof-of-Concept in the end.

On one side, the standard bandwidth that OAI and its community use is 40Mhz (106PRB) which requires +40MSps to send the band-base spectrum from the USRP to the host. And a single 1Gbps Ethernet connection can achieve up to 25MSps (Mega Samples per second) but it does not satisfy the network capacity of 61.44MSps that each USRP demands.

At the moment, the MCG Lab does not have a 10Gbps network to use (and even if it did have, the high bit-stream might saturate and delay others users' stream) nor the rack server has physical space for PCIe SPF+ cards for direct connection. The server in use is provisionally located at the Lab with a 1Gb connection but its intended location is in a server-room 4 floors above where, due to the +100m distance, the link is limited to 10Mb. For the future, it is recommended to create a dedicated 10 Gbps network for the USRPs to use in a USRP testbench rack.

It is possible to lower the number of PRBs to 24 (10Mhz bandwidth) in the gNB config file to lower the MSps network demand by changing the following parameters, but it would still need a 1Gb dedicated connection: `dl_carrierBandwidth`, `ul_carrierBandwidth`, `initialDLBWPlocationAndBandwidth` and `initialULBWPlocationAndBandwidth`.

On the other side, OAI requires a linux `low latency kernel` and direct access to the CPU governor to encode and decode LDCP frames in real-time and, although it is possible to set every configuration as if it was a physical machine, the underlying VM orchestrator still sets the CPU queues and priorities. For this, UHD generates many D and L errors regarding dropped frames or late packets in the network flow. To fix this, a more in-depth analysis of the VM environment should be done, which is out of the scope of this project.

This setup could have been used with RFSIM, but not with USRPs. As the initial intention was to use true radio heads, the migration to PC machines was done and this setup was not revisited.

6.2.3 OAI and FlexRIC

OAI and FlexRIC's lack of proper and up-to-date documentation and code commenting has been a constant struggle to be deal with during this project. Some MCG colleagues who had already worked with OAI shared these complaints. Therefore some works have switched to other software platforms such as srsRAN.

Many of the methods used in this project have been a result of searching through the code for options and information that were not documented anywhere else. In fact, the FlexRIC xApp SDK is not published together with the rest of the code in its GitLab repository but it is hidden away in its webpage as a downloadable zip file.

Additionally, a constant issue with OAI is to have developed tools for 4G but not to have them

functioning for 5G although it states the “nr” tag.

Moreover, the delay in FlexRIC’s new version (`dev`) release from the July to mid-August has made it impossible for me to test it due to lack of time. This new version adds many functionalities that would have been interesting to try out.

6.3 Future work

As for future lines work, I would consider the following:

- To deploy an End-to-End network with both the RAN and Core to use the SA mode instead of `phy-test`, establish a standard connection to send data through instead of the `noS1` mode and to try out the different RLC modes. Adding a Core network to this study opens many new possibilities.
- Explore the new FlexRIC version that uses a recent OAI version and adds new SMs and xApps.
- Try out the O-RAN RIC’s functionalities with the FlexRIC’s E2Agent to explore its possibilities.
- Utilize more powerful machines or an adequate server setup to eliminate errors due to lack of performance and open up the possibilities for higher throughput connections.

References

[1] ITU. *Recommendation ITU-R M.2083-0: IMT Vision - Framework and overall objectives of the future development of IMT for 2020 and beyond.* Sept. 2015. URL: https://www.itu.int/rec/R-REC-M.2083-0-201509-I/en.

[2] Emmanouil Pateromichelakis et al. "Service-Tailored User-Plane Design Framework and Architecture Considerations in 5G Radio Access Networks". In: *IEEE Access* PP (Aug. 2017), pp. 1–1. DOI: 10.1109/ACCESS.2017.2736579.

[3] 3GPP. *5G NR Overall description Stage-2 (3GPP TS 38.300 version 15.3.1 Release 15)*. Oct. 2018. URL: https://www.etsi.org/deliver/etsi_ts/138300_138399/138300/15.03.01_60/ts_138300v150301p.pdf.

[4] 3GPP. *5G NR Packet Data Convergence Protocol (PDCP) specification (3GPP TS 38.323 version 15.8.0 Release 15)*. Mar. 2021. URL: https://www.etsi.org/deliver/etsi_ts/138300_138399/138323/15.08.00_60/ts_138323v150800p.pdf.

[5] 3GPP. *UMTS Feasibility study for evolved Universal Terrestrial Radio Access (UTRA) and Universal Terrestrial Radio Access Network (UTRAN) (3GPP TR 25.912 version 7.2.0 Release 7)*. June 2007. URL: https://www.etsi.org/deliver/etsi_tr/125900_125999/125912/07.02.00_60/tr_125912v070200p.pdf.

[6] Techplayon.com. *5G NR SA Registration/Attach Call Flow*. URL: https://www.techplayon.com/5g-nr-sa-registration-attach-call-flow/.

[7] Larry Peterson and Oguz Sunay. *5G Mobile Networks: A Systems Approach*. 2022. URL: https://5g.systemsapproach.org/ran.html#split-ran.

[8] 5GWorldPro.com. *O-RAN introduced a specific category of split 7 called split 7-2x*. Feb. 2021. URL: https://www.5gworldpro.com/blog/2021/02/28/o-ran-introduced-a-specific-category-of-split-7-called-split-7-2x/.

[9] O-RAN Alliance. *About Us*. URL: https://www.o-ran.org/about.

[10] O-RAN Alliance. *O-RAN: Towards an Open and Smart RAN. White Paper*. Oct. 2018. URL: https://assets-global.website-files.com/60b4ffd4ca081979751b5ed2/60e5afb502810a0947b3b9d0_O-RAN%2BWP%2BFInal%2B181017.pdf.

[11] O-RAN Alliance. *O-RAN Near-RT RAN Intelligent Controller Architecture & E2 General Aspects and Principles 1.0*. ORAN-WG3.E2GAP-v01.00. Feb. 2020.

[12] Michele Polese et al. "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges". In: *CoRR* abs/2202.01032 (2022). URL: https://arxiv.org/abs/2202.01032.

[13] Mikel Irazabal Robert Schmidt and Navid Nikaein. “FlexRIC: An SDK for Next-Generation SD-RANs”. In: *The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT ’21), December 7–10, 2021, Virtual Event, Germany.* (Dec. 2021). URL: https://doi.org/10.1145/3485983.3494870.

[14] Open Air Interface. *Fall 2021 OpenAirInterface Workshop: Hands-On with the OAI Architects*. Dec. 2021. URL: https://openairinterface.org/fall-2021-openairinterface-workshop/.

[15] Eurecom. *FlexRIC GitLab’s Repository*. URL: https://gitlab.eurecom.fr/mosaic5g/flexric/-/tree/master/.

[16] Ettus. *Ettus Research Knowledge Base: USRP Host Performance Tuning Tips and Tricks*. URL: https://kb.ettus.com/USRP_Host_Performance_Tuning_Tips_and_Tricks. (accessed: 01.09.2022).

[17] Ettus. *Ettus Research Knowledge Base: USRP N300/N310*. URL: https://kb.ettus.com/N300/N310. (accessed: 01.09.2022).

[18] Ettus. *Ettus Research Knowledge Base: USRP N300/N310/N320/N321 Getting Started Guide*. URL: https://kb.ettus.com/USRP_N300/N310/N320/N321_Getting_Started_Guide. (accessed: 01.09.2022).

[19] Ettus. *USRP Manual: Transport Notes*. URL: https://files.ettus.com/manual/page_transport.html. (accessed: 01.09.2022).

[20] Ettus. *Ettus Research Knowledge Base: OctoClock CDA 2990*. URL: https://kb.ettus.com/OctoClock_CDA-2990/. (accessed: 01.09.2022).

[21] O-RAN Alliance. *O-RAN Near-Real-time RAN Intelligent Controller, E2 Application Protocol (E2AP) 1.0*. ORAN-WG3.E2AP-v01.00. Feb. 2020.

[22] Open Air Interface. *OpenAirInterface 5G RAN GitLab’s Repository*. URL: https://gitlab.eurecom.fr/oai/openairinterface5g.

[23] Wireshark. URL: https://www.wireshark.org/.

[24] Rohde&Schwarz. *R&S FSW Signal and spectrum analyzer*. URL: https://www.rohde-schwarz.com/es/productos/test-y-medida/analizadores-de-sobremesa/rs-fsw_63493-11793.html.

[25] Rohde&Schwarz. *R&S ROMES4 Drive test software*. URL: https://www.rohde-schwarz.com/es/productos/test-y-medida/recopilacion-de-datos-de-red/rs-romes4_63493-8650.html.

[26] Iperf. URL: https://iperf.fr/iperf-doc.php.

[27] Sai Shruthi Nakkina, Sudhakar Balijepalli, and Chandra R. Murthy. “Performance Benchmarking of the 5G NR PHY on the OAI Codebase and USRP Hardware”. In: *WSA 2021; 25th International ITG Workshop on Smart Antennas*. 2021, pp. 1–6. URL: https://ieeexplore.ieee.org/document/9739187.

[28] 3GPP. *5G NR User Equipment (UE) radio access capabilities (3GPP TS 38.306 version 16.3.0 Release 16)*. Jan. 2021. URL: https://www.etsi.org/deliver/etsi_ts/138300_138399/138306/16.03.00_60/ts_138306v160300p.pdf.

[29] 3GPP. *5G NR Physical layer procedures for data (3GPP TS 38.214 version 15.16.0 Release 15)*. Apr. 2022. URL: https://www.etsi.org/deliver/etsi_ts/138200_138299/138214/15.16.00_60/ts_138214v151600p.pdf.

[30] 5G-tools.com. *5G NR TBS (Transport Block size) Calculator*. URL: https://5g-tools.com/5g-nr-tbs-transport-block-size-calculator/. (accessed: 01.09.2022).

[31] Matplotlib. *FAQ: Working with threads*. URL: https://matplotlib.org/3.1.0/faq/howto_faq.html#working-with-threads.

[32] Tammo Ippen. *Plotille's GitHub repository*. URL: https://github.com/tammoippen/plotille.

Part II

Annexes

Appendix A

gNB configuration file

Listing A.1: gnb.band78.tm1.fr1.106PRB.usrpn310.conf

```
Active_gNBs = ( "gNB-OAI");
Asn1_verbosity = "none";

gNBs = ( {
    ////////// Identification parameters:
    gNB_CU_ID = 0xe00;
    gNB_ID = 0xe00;
    gNB_name  =  "gNB-OAI";
    // Tracking area code, 0x0000 and 0xfffe are reserved values
    tracking_area_code  =  1;
    plmn_list = ({
                  mcc = 208;
                  mnc = 99;
                  mnc_length = 2;
                  snssaiList = ( {
                                   sst = 1;
                                   sd  = 0x1; // 0 false, else true
                                 },
                                 {
                                   sst = 1;
                                   sd  = 0x112233; // 0 false, else true
                                 });
                });
    nr_cellid = 12345678L

    ////////// Physical parameters:
    ssb_SubcarrierOffset                                    = 0;
    pdsch_AntennaPorts                                      = 1;
    pusch_AntennaPorts                                      = 1;
    sib1_tda                                                = 1;
    min_rxtxtime_pdsch = 6;
    min_rxtxtime = 6;
        do_SRS = 1;

    pdcch_ConfigSIB1 = ( {
        controlResourceSetZero = 11;#11
        searchSpaceZero = 0;
```

```
      } );

    servingCellConfigCommon = ({     #spCellConfigCommon
      physCellId                                                    = 0;
#  downlinkConfigCommon
    #frequencyInfoDL
      # this is 3300.60 MHz + 53*12*30e-3 MHz = 3319.68
      absoluteFrequencySSB                                          =
          621312;
      dl_frequencyBand                                              = 78;
      # this is 3300.60 MHz
      dl_absoluteFrequencyPointA                                    =
          620040;
      #scs-SpecificCarrierList
        dl_offstToCarrier                                           = 0;
# subcarrierSpacing
# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
        dl_subcarrierSpacing                                        = 1;
        dl_carrierBandwidth                                         = 106;
     #initialDownlinkBWP
      #genericParameters
        # this is RBstart=0,L=106 (275*(L-1))+RBstart
        initialDLBWPlocationAndBandwidth                            = 28875;
# subcarrierSpacing
# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
        initialDLBWPsubcarrierSpacing                               = 1;
      #pdcch-ConfigCommon
        initialDLBWPcontrolResourceSetZero                          = 11;
        initialDLBWPsearchSpaceZero                                 = 0;
      #pdsch-ConfigCommon
        #pdschTimeDomainAllocationList (up to 16 entries)
        initialDLBWPk0_0                        = 0;  #for DL slot
        initialDLBWPmappingType_0               = 0;  #0=typeA,1=typeB
        initialDLBWPstartSymbolAndLength_0      = 40; #this is SS=1,L=13
        initialDLBWPk0_1                        = 0;  #for mixed slot
        initialDLBWPmappingType_1               = 0;
        initialDLBWPstartSymbolAndLength_1      = 57; #this is SS=1,L=5

  #uplinkConfigCommon
     #frequencyInfoUL
      ul_frequencyBand                                                = 78;
      #scs-SpecificCarrierList
      ul_offstToCarrier                                               = 0;
# subcarrierSpacing
# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
      ul_subcarrierSpacing                                          = 1;
      ul_carrierBandwidth                                           = 106;
      pMax                                                          = 20;
     #initialUplinkBWP
      #genericParameters
        initialULBWPlocationAndBandwidth                            = 28875;
# subcarrierSpacing
# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
        initialULBWPsubcarrierSpacing                               = 1;
      #rach-ConfigCommon
        #rach-ConfigGeneric
          prach_ConfigurationIndex                                  = 98;
#prach_msg1_FDM
```

```
#0 = one, 1=two, 2=four, 3=eight
          prach_msg1_FDM                                          = 0;
          prach_msg1_FrequencyStart                               = 0;
          zeroCorrelationZoneConfig                               = 12;
          preambleReceivedTargetPower                             = -96;
#preamblTransMax (0...10) = (3,4,5,6,7,8,10,20,50,100,200)
          preambleTransMax                                        = 6;
#powerRampingStep
# 0=dB0,1=dB2,2=dB4,3=dB6
        powerRampingStep                                          = 1;
#ra_ReponseWindow
#1,2,4,8,10,20,40,80
        ra_ResponseWindow                                         = 4;
#ssb_perRACH_OccasionAndCB_PreamblesPerSSB_PR
#1=oneeighth,2=onefourth,3=half,4=one,5=two,6=four,7=eight,8=sixteen
        ssb_perRACH_OccasionAndCB_PreamblesPerSSB_PR              = 3;
#oneHalf (0..15) 4,8,12,16,...60,64
        ssb_perRACH_OccasionAndCB_PreamblesPerSSB                 = 15;
#ra_ContentionResolutionTimer
#(0..7) 8,16,24,32,40,48,56,64
        ra_ContentionResolutionTimer                              = 7;
        rsrp_ThresholdSSB                                         = 19;
#prach-RootSequenceIndex_PR
#1 = 839, 2 = 139
        prach_RootSequenceIndex_PR                                = 2;
        prach_RootSequenceIndex                                   = 1;
        # SCS for msg1, can only be 15 for 30 kHz < 6 GHz, takes precendence
            over the one derived from prach-ConfigIndex
        #
        msg1_SubcarrierSpacing                                    = 1,
# restrictedSetConfig
# 0=unrestricted, 1=restricted type A, 2=restricted type B
        restrictedSetConfig                                       = 0,
      # pusch-ConfigCommon (up to 16 elements)
        initialULBWPk2_0                          = 6;  # used for UL slot
        initialULBWPmappingType_0                 = 1
        initialULBWPstartSymbolAndLength_0        = 41; # this is SS=0 L=13

        initialULBWPk2_1                          = 6;  # used for mixed slot
        initialULBWPmappingType_1                 = 1;
        initialULBWPstartSymbolAndLength_1        = 52; # this is SS=10 L=4

        initialULBWPk2_2                          = 7;  # used for Msg.3 during RA
        initialULBWPmappingType_2                 = 1;
        initialULBWPstartSymbolAndLength_2        = 52; # this is SS=10 L=4

        msg3_DeltaPreamble                                        = 1;
        p0_NominalWithGrant                                       =-90;

# pucch-ConfigCommon setup :
# pucchGroupHopping
# 0 = neither, 1= group hopping, 2=sequence hopping
        pucchGroupHopping                                         = 0;
        hoppingId                                                 = 40;
        p0_nominal                                                = -90;
# ssb_PositionsInBurs_BitmapPR
# 1=short, 2=medium, 3=long
      ssb_PositionsInBurst_PR                                     = 2;
```

```
      ssb_PositionsInBurst_Bitmap                                   = 1;

# ssb_periodicityServingCell
# 0 = ms5, 1=ms10, 2=ms20, 3=ms40, 4=ms80, 5=ms160, 6=spare2, 7=spare1
      ssb_periodicityServingCell                                    = 2;

# dmrs_TypeA_position
# 0 = pos2, 1 = pos3
      dmrs_TypeA_Position                                           = 0;

# subcarrierSpacing
# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
      subcarrierSpacing                                             = 1;


  #tdd-UL-DL-ConfigurationCommon
# subcarrierSpacing
# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
      referenceSubcarrierSpacing                                    = 1;
      # pattern1
      # dl_UL_TransmissionPeriodicity
      # 0=ms0p5, 1=ms0p625, 2=ms1, 3=ms1p25, 4=ms2, 5=ms2p5, 6=ms5, 7=ms10
      dl_UL_TransmissionPeriodicity                                 = 6;
      nrofDownlinkSlots                                             = 7;
      nrofDownlinkSymbols                                           = 6;
      nrofUplinkSlots                                               = 2;
      nrofUplinkSymbols                                             = 4;

  ssPBCH_BlockPower                                               = -25;
  }

  );

# Dedicated Serving Cell Configuration
servingCellConfigDedicated = ({
  # BWP-Downlink
    # BWP 1 Configuration
      dl_bwp-Id_1 = 1;
      dl_bwp1_locationAndBandwidth = 28875; // RBstart=0, L=106 (40 MHz BW)
      # subcarrierSpacing
      # 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
      dl_bwp1_subcarrierSpacing = 1;

    firstActiveDownlinkBWP-Id = 1;  #BWP-Id
    defaultDownlinkBWP-Id     = 1;  #BWP-Id

  # UplinkConfig
    # BWP-Uplink
      # BWP 1 Configuration
        ul_bwp-Id_1 = 1;
        ul_bwp1_locationAndBandwidth = 28875; // RBstart=0, L=106 (40 MHz BW)
        # subcarrierSpacing
        # 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
        ul_bwp1_subcarrierSpacing = 1;

      firstActiveUplinkBWP-Id = 1;  #BWP-Id
  }
);
```

```
    # ------- SCTP definitions
    SCTP : {
        # Number of streams to use in input/output
        SCTP_INSTREAMS  = 2;
        SCTP_OUTSTREAMS = 2;
    };
    ////////// AMF parameters:
        amf_ip_address      = ( { ipv4       = "192.168.71.132";
                                  ipv6       = "192:168:30::17";
                                  active     = "yes";
                                  preference = "ipv4"; } );
        NETWORK_INTERFACES : {
            GNB_INTERFACE_NAME_FOR_NG_AMF            = "rfsim5g-public";
            GNB_IPV4_ADDRESS_FOR_NG_AMF              = "192.168.71.129";
            GNB_INTERFACE_NAME_FOR_NGU               = "rfsim5g-public";
            GNB_IPV4_ADDRESS_FOR_NGU                 = "192.168.71.129";
            GNB_PORT_FOR_S1U                         = 2152; # Spec 2152
        };
  } );

MACRLCs = ( {
    num_cc           = 1;
    tr_s_preference  = "local_L1";
    tr_n_preference  = "local_RRC";
    pusch_TargetSNRx10 = 200;
    pucch_TargetSNRx10 = 200;
     ulsch_max_frame_inactivity = 0; } );

L1s = ( {
      num_cc = 1;
      tr_n_preference = "local_mac";
      pusch_proc_threads = 4;
      prach_dtx_threshold = 120;
      pucch0_dtx_threshold = 150;
        ofdm_offset_divisor   = 8;
      } );

RUs = ( {
       local_rf       = "yes"
         nb_tx          = 1
         nb_rx          = 1
         att_tx         = 0
         att_rx         = 0;
         bands          = [78];
         max_pdschReferenceSignalPower = -27;
         max_rxgain                    = 75;
         eNB_instances  = [0];
         ##beamforming 1x2 matrix: 1 layer x 2 antennas
         bf_weights = [0x00007fff, 0x0000];
         clock_src = "external";
         sdr_addrs = "addr=192.168.10.2,clock_source=external,time_source=
             external"
    } );

THREAD_STRUCT = ( {
    #three config for level of parallelism "PARALLEL_SINGLE_THREAD", "
        PARALLEL_RU_L1_SPLIT", or "PARALLEL_RU_L1_TRX_SPLIT"
```

```
    parallel_config     = "PARALLEL_SINGLE_THREAD";
    #two option for worker "WORKER_DISABLE" or "WORKER_ENABLE"
    worker_config       = "WORKER_ENABLE";
  } );

security = {
  # preferred ciphering algorithms
  # the first one of the list that an UE supports in chosen
  # valid values: nea0, nea1, nea2, nea3
  ciphering_algorithms = ( "nea0" );

  # preferred integrity algorithms
  # the first one of the list that an UE supports in chosen
  # valid values: nia0, nia1, nia2, nia3
  integrity_algorithms = ( "nia2", "nia0" );

  # setting 'drb_ciphering' to "no" disables ciphering for DRBs, no matter
  # what 'ciphering_algorithms' configures; same thing for 'drb_integrity'
  drb_ciphering = "yes";
  drb_integrity = "no";
};

     log_config :
     {
       global_log_level                      ="info";
       global_log_verbosity                  ="medium";
       hw_log_level                          ="info";
       hw_log_verbosity                      ="medium";
       phy_log_level                         ="info";
       phy_log_verbosity                     ="low";
       mac_log_level                         ="info";
       mac_log_verbosity                     ="high";
       rlc_log_level                         ="info";
       rlc_log_verbosity                     ="medium";
       pdcp_log_level                        ="info";
       pdcp_log_verbosity                    ="medium";
       rrc_log_level                         ="info";
       rrc_log_verbosity                     ="medium";
       f1ap_log_level                         ="debug";
       f1ap_log_verbosity                     ="medium";
    };
```

Appendix B

E2AP available stats list

- **MAC layer**
  - `mac.dl_aggr_tbs` Downlink aggregate transport block size.
  - `mac.ul_aggr_tbs` Uplink aggregate transport block
  - `mac.dl_aggr_sdus` Downlink aggregate sdu
  - `mac.ul_aggr_sdus` Uplink aggregate sdu
  - `mac.dl_aggr_bytes_sdus` Downlink aggregate bytes sdus
  - `mac.ul_aggr_bytes_sdus` Uplink aggregate bytes sdus
  - `mac.dl_aggr_prb` Downlink aggregate physical resource block
  - `mac.ul_aggr_prb` Uplink aggregate physical resource block
  - `mac.dl_aggr_retx_prb` Downlink aggregate re-transmission physical resource block
  - `mac.wb_cqi` Channel quality indicator
  - `mac.dl_mcs1` Downlink modulation and coding scheme 1
  - `mac.ul_mcs1` Uplink modulation and coding scheme 1
  - `mac.dl_mcs2` Downlink modulation and coding scheme 2
  - `mac.ul_mcs2` Uplink modulation and coding scheme 2
  - `mac.phr` Power head room
  - `mac.pusch_snr` Pusch snr
  - `mac.pucch_snr` Pucch snr
  - `mac.rnti` Radio network temporary identifier
- **RLC layer**
  - `rlc.txpdu_pkts` Aggregated number of transmitted RLC PDUs
  - `rlc.txpdu_bytes` Aggregated amount of transmitted bytes in RLC PDUs
  - `rlc.txpdu_wt_ms` Aggregated head-of-line tx packet waiting time to be transmitted (i.e. send to the MAC layer)

- `rlc.txpdu_dd_pkts` Aggregated number of dropped or discarded tx packets by RLC
- `rlc.txpdu_dd_bytes` Aggregated amount of bytes dropped or discarded tx packets by RLC
- `rlc.txpdu_retx_pkts` Aggregated number of tx pdus/pkts to be re-transmitted (only applicable to RLC AM)
- `rlc.txpdu_retx_bytes` Aggregated amount of bytes to be re-transmitted (only applicable to RLC AM)
- `rlc.txpdu_segmented` Aggregated number of segmentations
- `rlc.txpdu_status_pkts` Aggregated number of tx status pdus/pkts (only applicable to RLC AM)
- `rlc.txpdu_status_bytes` Aggregated amount of tx status bytes (only applicable to RLC AM)
- `rlc.txbuf_occ_bytes` Current tx buffer occupancy in terms of amount of bytes (average: NOT IMPLEMENTED)
- `rlc.txbuf_occ_pkts` Current tx buffer occupancy in terms of number of packets (average: NOT IMPLEMENTED)
- `rlc.rxpdu_pkts` Aggregated number of received RLC PDUs
- `rlc.rxpdu_bytes` Amount of bytes received by the RLC
- `rlc.rxpdu_dup_pkts` Aggregated number of duplicate packets
- `rlc.rxpdu_dup_bytes` Aggregated amount of duplicated bytes
- `rlc.rxpdu_dd_pkts` Aggregated number of rx packets dropped or discarded by RLC
- `rlc.rxpdu_dd_bytes` Aggregated amount of rx bytes dropped or discarded by RLC
- `rlc.rxpdu_ow_pkts` Aggregated number of out of window received RLC pdu
- `rlc.rxpdu_ow_bytes` Aggregated number of out of window bytes received RLC pdu
- `rlc.rxpdu_status_pkts` Aggregated number of rx status pdus/pkts (only applicable to RLC AM)
- `rlc.rxpdu_status_bytes` Aggregated amount of rx status bytes (only applicable to RLC AM)
- `rlc.rxbuf_occ_bytes` Current rx buffer occupancy in terms of amount of bytes (average: NOT IMPLEMENTED)
- `rlc.rxbuf_occ_pkts` Current rx buffer occupancy in terms of number of packets (average: NOT IMPLEMENTED)
- `rlc.txsdu_pkts` Number of SDUs delivered
- `rlc.txsdu_bytes` Number of bytes of SDUs delivered
- `rlc.rxsdu_pkts` Number of SDUs received
- `rlc.rxsdu_bytes` Number of bytes of SDUs received

- `rlc.rxsdu_dd_pkts` Number of dropped or discarded SDUs
- `rlc.rxsdu_dd_bytes` Number of bytes of SDUs dropped or discarded
- `rlc.rnti` Radio Network Temporary Identifier
- `rlc.mode` 0: RLC Ack Mode, 1: RLC Unack Mode, 2: RLC Transparent Mode
- `rlc.rbid` Radio bearer id
- `rlc.frame` Radio frame
- `rlc.slot` Radio slot

- **PDCP layer**
  - `pdcp.txpdu_pkts` Aggregated number of tx packets
  - `pdcp.txpdu_bytes` Aggregated bytes of tx packets
  - `pdcp.txpdu_sn` Current sequence number of last tx packet (or `TX_NEXT`)
  - `pdcp.rxpdu_pkts` Aggregated number of rx packets
  - `pdcp.rxpdu_bytes` Aggregated bytes of rx packets
  - `pdcp.rxpdu_sn` Current sequence number of last rx packet (or `RX_NEXT`)
  - `pdcp.rxpdu_oo_pkts` Aggregated number of out-of-order rx pkts (or `RX_REORD`)
  - `pdcp.rxpdu_oo_bytes` Aggregated amount of out-of-order rx bytes
  - `pdcp.rxpdu_dd_pkts` Aggregated number of duplicated discarded packets (or dropped packets because of other reasons such as integrity failure) (or `RX_DELIV`)
  - `pdcp.rxpdu_dd_bytes` Aggregated amount of discarded packets' bytes
  - `pdcp.rxpdu_ro_count` This state variable indicates the COUNT value following the COUNT value associated with the PDCP Data PDU which triggered t-Reordering. (`RX_REORD`)
  - `pdcp.txsdu_pkts` Number of SDUs delivered
  - `pdcp.txsdu_bytes` Number of bytes of SDUs delivered
  - `pdcp.rxsdu_pkts` Number of SDUs received
  - `pdcp.rxsdu_bytes` Number of bytes of SDUs received
  - `pdcp.rnti` Radio Network Temporary Identifier
  - `pdcp.mode` Mode of pdcp
  - `pdcp.rbid` Radio bearer
  - `pdcp.tstamp` Time stamp

Appendix C

Throughput xApp

```
# The 3-Clause BSD License
# Redistribution and use in source and binary forms, with or without modification, are permitted provided that the
      following conditions are met:
# 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following
      disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the
      following disclaimer in the documentation and/or other materials provided with the distribution.
# 3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote
      products derived from this software without specific prior written permission.

# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
# WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
# PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
# ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
# TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
# NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
# THE POSSIBILITY OF SUCH DAMAGE.

## Authors: Hung NGUYEN, OAI M5G
##          Sergio MALLASÉN, MCG iTeam UPV

import sys
sys.path.append('./src')
import logging

from sdk.xapp_base import Xappbase
from sdk.constants import Ind
import asyncio
import sdk.comm_layer as comm_layer
from sdk.utils import do_nothing, pdcp_attr, mac_attr, rlc_attr
import sdk.api as api
from sdk import context

import plotille
import collections

class Xapp(Xappbase):
    def __init__(self, conf):
        period_to_run_logic = 1 - seconds -

        super().__init__(
            period_to_run_logic,
            log_level=logging.DEBUG,
            ping_timeout=conf['connection_timeout'],
            record = False,
            simulation = False,
            trigger_periodic_func=True
            )

        # subscribe for the services
        self.report(Ind.MAC)
        self.report(Ind.RLC)
        self.report(Ind.PDCP)

        # Variables for holding stats data
        self.mac_timestamps = list()
        self.rlc_timestamps = list()
        self.pdcp_timestamps = list()

        self.mac_delta = collections.deque(maxlen=60)
```

```
        self.rlc_delta = collections.deque(maxlen=60)
        self.pdcp_delta = collections.deque(maxlen=60)

        self.mac_dl_aggr_tbs = list()
        self.mac_dl_aggr_tbs.append(0)
        self.mac_ul_aggr_tbs = list()
        self.mac_ul_aggr_tbs.append(0)

        self.rlc_rxpdu_bytes = list()
        self.rlc_rxpdu_bytes.append(0)
        self.rlc_txpdu_bytes = list()
        self.rlc_txpdu_bytes.append(0)

        self.pdcp_rxpdu_bytes = list()
        self.pdcp_rxpdu_bytes.append(0)
        self.pdcp_txpdu_bytes = list()
        self.pdcp_txpdu_bytes.append(0)

        self.mac_dl_throughput = collections.deque(maxlen=60)
        self.mac_ul_throughput = collections.deque(maxlen=60)

        self.rlc_rx_throughput = collections.deque(maxlen=60)
        self.rlc_tx_throughput = collections.deque(maxlen=60)

        self.pdcp_rx_throughput = collections.deque(maxlen=60)
        self.pdcp_tx_throughput = collections.deque(maxlen=60)

        self.flag_first_time = 1

    def run_periodic(self, *args, **kwargs):
        - This function will call every self.period seconds to the logic. -
        if api.number_connected_ue():
            fig = plotille.Figure()            - Set fig -
            fig.set_y_limits(min_=0)
            fig.x_label="seconds"
            fig.y_label="Mbytes"
            fig.x_ticks_fkt = str_tick          -Ticks with 3 decimal places -
            fig.y_ticks_fkt = str_tick
            fig.width = 105
            fig.height = 33

            pdcp_stats = api.get_attr(attr=["pdcp.rnti", "pdcp.tstamp", "pdcp.txpdu_bytes", "pdcp.rxpdu_bytes"],
                 windows=1)
            self.pdcp_timestamps.append(pdcp_stats[0][0]['pdcp.tstamp'][0])
            self.pdcp_rxpdu_bytes.append(pdcp_stats[0][0]['pdcp.rxpdu_bytes'][0])
            self.pdcp_txpdu_bytes.append(pdcp_stats[0][0]['pdcp.txpdu_bytes'][0])

            rlc_stats = api.get_attr(attr=["rlc.rnti", "rlc.tstamp", "rlc.txpdu_bytes", "rlc.rxpdu_bytes"], windows=1)
            self.rlc_timestamps.append(rlc_stats[0][0]['rlc.tstamp'][0])
            self.rlc_rxpdu_bytes.append(rlc_stats[0][0]['rlc.rxpdu_bytes'][0])
            self.rlc_txpdu_bytes.append(rlc_stats[0][0]['rlc.txpdu_bytes'][0])

            mac_stats = api.get_attr(attr=["mac.rnti", "mac.tstamp", "mac.dl_aggr_tbs", "mac.ul_aggr_tbs"], windows=1)
            self.mac_timestamps.append(mac_stats[0][0]['mac.tstamp'][0])
            self.mac_dl_aggr_tbs.append(mac_stats[0][0]['mac.dl_aggr_tbs'][0])
            self.mac_ul_aggr_tbs.append(mac_stats[0][0]['mac.ul_aggr_tbs'][0])

            self.pdcp_delta.append((self.pdcp_timestamps[-1]-self.pdcp_timestamps[0]) / 1e6)
            self.rlc_delta.append((self.rlc_timestamps[-1]-self.rlc_timestamps[0]) / 1e6)
            self.mac_delta.append((self.mac_timestamps[-1]-self.mac_timestamps[0]) / 1e6)

            if self.flag_first_time == 1:
                self.flag_first_time = 0
                self.pdcp_rxpdu_bytes.append(pdcp_stats[0][0]['pdcp.rxpdu_bytes'][0])
                self.pdcp_txpdu_bytes.append(pdcp_stats[0][0]['pdcp.txpdu_bytes'][0])
                self.rlc_rxpdu_bytes.append(rlc_stats[0][0]['rlc.rxpdu_bytes'][0])
                self.rlc_txpdu_bytes.append(rlc_stats[0][0]['rlc.txpdu_bytes'][0])
                self.mac_dl_aggr_tbs.append(mac_stats[0][0]['mac.dl_aggr_tbs'][0])
                self.mac_ul_aggr_tbs.append(mac_stats[0][0]['mac.ul_aggr_tbs'][0])

                self.pdcp_rx_throughput.append(0)
                self.pdcp_tx_throughput.append(0)

                self.rlc_rx_throughput.append(0)
                self.rlc_tx_throughput.append(0)

                self.mac_ul_throughput.append(0)
                self.mac_dl_throughput.append(0)
            else:
                self.pdcp_rx_throughput.append(((self.pdcp_rxpdu_bytes[-1]-self.pdcp_rxpdu_bytes[-2])*8 / 1e6) / (self
                     .pdcp_delta[-1]-self.pdcp_delta[-2]))
                self.pdcp_tx_throughput.append(((self.pdcp_txpdu_bytes[-1]-self.pdcp_txpdu_bytes[-2])*8 / 1e6) / (self
                     .pdcp_delta[-1]-self.pdcp_delta[-2]))

                self.rlc_rx_throughput.append(((self.rlc_rxpdu_bytes[-1]-self.rlc_rxpdu_bytes[-2])*8 / 1e6) / (self.
                     rlc_delta[-1]-self.rlc_delta[-2]))
                self.rlc_tx_throughput.append(((self.rlc_txpdu_bytes[-1]-self.rlc_txpdu_bytes[-2])*8 / 1e6) / (self.
                     rlc_delta[-1]-self.rlc_delta[-2]))

                self.mac_dl_throughput.append(((self.mac_dl_aggr_tbs[-1]-self.mac_dl_aggr_tbs[-2])*8 / 1e6) / (self.
                     mac_delta[-1]-self.mac_delta[-2]))
```

```
                self.mac_ul_throughput.append(((self.mac_ul_aggr_tbs[-1]-self.mac_ul_aggr_tbs[-2])*8 / 1e6) / (self.
                    mac_delta[-1]-self.mac_delta[-2]))

            fig.plot(self.pdcp_delta, self.pdcp_rx_throughput, label="PDCP Rx PDU throughput", lc="blue")
            fig.plot(self.pdcp_delta, self.pdcp_tx_throughput, label="PDCP Tx PDU throughput", lc="cyan")

            fig.plot(self.rlc_delta, self.rlc_rx_throughput, label="RLC Rx PDU throughput", lc="red")
            fig.plot(self.rlc_delta, self.rlc_tx_throughput, label="RLC Tx PDU throughput", lc="magenta")

            fig.plot(self.mac_delta, self.mac_ul_throughput, label="MAC UL throughput", lc="white")
            fig.plot(self.mac_delta, self.mac_dl_throughput, label="MAC DL throughput", lc="yellow")

            print(fig.show(legend=True))

            #if not self.simulation:
                ## This stats are only valid when we run with real RIC
                # Not on the simulation mode.
                #self.log.info(f'Number of Mac msg received recv so far: {self.mac_recv}')
                #self.log.info(f'Number of RLC msg received recv so far : {self.rlc_recv}')
                #self.log.info(f'Number of PDCP msg received recv so far : {self.pdcp_recv}')

        else:
            self.log.info("No ue connected")

def str_tick(min_, max_):           - To show axes values with only 3 decimals -
    return '{:.3f}'.format(min_ + (max_-min_) / 2)

async def main():
    conf = {
        'connection_timeout' : 1,
        'running_time': 240,
    }
    xapp = Xapp(conf)
    await xapp.run()
    await asyncio.sleep(conf["running_time"])
    xapp.stop()

if __name__ == "__main__":
    asyncio.run(main())
```